PATENT ABSTRACTS OF JAPAN

(11)Publication number : 09-016168
(43)Date of publication of application : 17.01.1997

(51)Int.CI. G10H 1/00
G10H 1/00
H03M 7/46

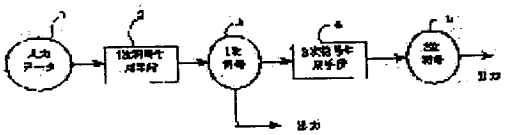
(21)Application number : 07-306780
(22)Date of filing : 30.10.1995

(71)Applicant : VICTOR CO OF JAPAN LTD
(72)Inventor : SHISHIDO ICHIRO

(30)Priority
Priority number : 07129017 Priority date : 28.04.1995 Priority country : JP

(54) COMPRESSING DEVICE AND DECODING DEVICE FOR MUSICAL PERFORMANCE INFORMATION

(57)Abstract:
PROBLEM TO BE SOLVED: To efficiently compress and expand the data of the musical performance information by an LZ method.
SOLUTION: Input data 1 in SMF format are analyzed by a primary code generating means 2 and the musical performance information is separated into at least an interval, strength, length, and other information to generate a primary code 3 having respective pieces of information arranged in independent areas. The codes of the respective areas of the primary code 3 are compressed by the LZ method through a secondary code generating means 4 to generate a secondary code 5. The primary code generating means 2 rearranges six kinds of primary compressed codes, i.e., a note A code indicating one note, a controller „ code, a duration code, a node number code, a velocity code, and a controller code by a code arranging means and outputs them as the primary code 3 to the secondary code generating means 4, which performs secondary compression. The data after the secondary compression are processed through the secondary decoding of a secondary code decoding means 23 and then a primary code decoding means performs primary decoding.



LEGAL STATUS

[Date of request for examination]	30.09.1998
[Date of sending the examiner's decision of rejection]	
[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]	
[Date of final disposal for application]	
[Patent number]	3246301
[Date of registration]	02.11.2001
[Number of appeal against examiner's decision of rejection]	
[Date of requesting appeal against examiner's decision of rejection]	
[Date of extinction of right]	

(書誌＋要約＋請求の範囲)

(19)【発行国】日本国特許庁(JP)
(12)【公報種別】公開特許公報(A)
(11)【公開番号】特開平9－16168
(43)【公開日】平成9年(1997)1月17日
(54)【発明の名称】演奏情報圧縮装置及び演奏情報復号装置
(51)【国際特許分類第6版】

G10H 1/00
102
H03M 7/46

【FI】

G10H 1/00 Z
102 Z
H03M 7/46 9382-5K

【審査請求】未請求
【請求項の数】9
【出願形態】FD
【全頁数】18
(21)【出願番号】特願平7－306780
(22)【出願日】平成7年(1995)10月30日
(31)【優先権主張番号】特願平7－129017
(32)【優先日】平7(1995)4月28日
(33)【優先権主張国】日本(JP)
(71)【出願人】
【識別番号】000004329
【氏名又は名称】日本ビクター株式会社
【住所又は居所】神奈川県横浜市神奈川区守屋町3丁目12番地
(72)【発明者】
【氏名】宍戸 一郎
【住所又は居所】神奈川県横浜市神奈川区守屋町3丁目12番地 日本ビクター株式会社内
(74)【代理人】
【弁理士】
【氏名又は名称】二瓶 正敬

---

(57)【要約】
【課題】LZ法により演奏情報のデータ量を効率的に圧縮し、伸長する。
【解決手段】SMFフォーマットの入力データ1は1次符号生成手段2により解析され、演奏情報が少なくとも音程と、強さと、長さとその他の情報に分離され、各情報がそれぞれ独立した領域に配置した1次符号3が生成される。この1次符号3の各領域の符号は2次符号生成手段4によりLZ法で圧縮され、2次符号5が生成される。また、1次符号生成手段2では1つの音符を示すノート„ 符号、コントローラ„ 符号、デュレーション符号、ノートナンバ符号、ベロシティ符号及びコントローラ符号の6種類の1次圧縮符号が符号配置手段19により並べ換えられ、1次符号3として2次符号生成手段4に出力され、2次符号生成手段4により2次圧縮される。2次圧縮されたデータは2次符号復号手段23により2次復号され、次いで1次符号復号手段24により1次復号される。

【特許請求の範囲】
【請求項1】演奏情報を少なくとも音程の情報と、音の強さの情報と、音の長さの情報と、その他の情報に分離し、前記各情報をそれぞれ独立した領域に配置した1次符号を生成する1次符号生成手段と、前記1次符号生成手段により生成された1次符号の各領域の情報をLZ法により圧縮する2次符号生成手段とを有する演奏情報圧縮装置。
【請求項2】前記1次符号生成手段は、各イベント間の相対時間の公約数及び各音符の長さの公約数を算出し、各イベント間の相対時間及び各音符の長さの値をこれらの公約数で除算した後符号化することを特徴とする請求項1記載の演奏情報圧縮装置。
【請求項3】前記1次符号生成手段は、各音符の音程情報をそれ以前に出現した音符の音程の数値を使って一定の関数式に従って算出した数値と実際の音程の数値との残差で表すことを特徴とする請求項1又は2記載の演奏情報圧縮装置。
【請求項4】前記1次符号生成手段は、各音符の強さ情報をそれ以前に出現した音符の強さの数値を使って一定の

関数式に従って算出した数値と実際の強さの数値との残差で表すことを特徴とする請求項1乃至3のいずれか一つに記載の演奏情報圧縮装置。
【請求項5】前記1次符号生成手段は、特定の種類のイベントのパラメータ値をそれ以前に出現した同種類のイベントのパラメータ値を使って一定の関数式に従って算出した数値と実際のパラメータ値との残差で表すことを特徴とする請求項1乃至4のいずれか一つに記載の演奏情報圧縮装置。
【請求項6】前記1次符号生成手段は、前記各情報を当該領域においてトラック順に配置したことをことを特徴とする請求項1乃至5のいずれか一つに記載の演奏情報圧縮装置。
【請求項7】前記1次符号生成手段は、前記各領域をデータの性質が似ている領域同志が近くなるように配置したことを特徴とする請求項6記載の演奏情報圧縮装置。
【請求項8】演奏情報を少なくとも音程の情報と、音の強さの情報と、音の長さの情報と、その他の情報に分離し、前記各情報をそれぞれ独立した領域に配置した1次符号を復号する演奏情報復号装置であって、供給される前記1次符号を演奏情報に復号する1次符号復号手段を有することを特徴とする演奏情報復号装置。
【請求項9】演奏情報を少なくとも音程の情報と、音の強さの情報と、音の長さの情報と、その他の情報に分離し、前記各情報をそれぞれ独立した領域に配置した1次符号を生成し、この1次符号の各領域の情報をLZ法により圧縮した2次符号を復号する演奏情報復号装置であって、前記2次符号を前記1次符号に復号する2次符号復号手段と、前記2次符号復号手段により復号された1次符号を演奏情報に復号する1次符号復号手段とを、有することを特徴とする演奏情報復号装置。

詳細な説明

【発明の詳細な説明】
【0001】
【発明の属する技術分野】本発明は、演奏情報のデータ量を圧縮する演奏情報圧縮装置及び演奏情報の圧縮データを復号する演奏情報復号装置に関する。
【0002】
【従来の技術】一般に、MIDI(Musical Instrument Digital Interface)データを保存する方式として、スタンダードMIDIファイル(以下、SMFという。)が広く用いられ、このSMFでは図19に示すように個々の演奏情報がデルタ(„)タイムとイベントの2つの要素により構成されている。„タイムは、隣合ったイベント間の相対時間を表し、イベントはノートオン又はノートオフのステータス、音程(ノートナンバ)や音の強さ(ベロシティ)などの種々の演奏情報を含む。ここで、演奏情報とは音符により示される音程、音の長さの他に、音の強さ、楽曲の演奏上の拍子、テンポなどの情報、さらに音源の種類、リセットコントロールの情報などを含むものを指すものとする。また、このSMFでは各演奏情報が時間順に並んでトラックを構成している。ここで、SMF方式は記憶容量や伝送路の効率的利用という点ではあまり優れたものではないので、通信カラオケや音楽データベースのように多数の楽曲データを記録、伝送するシステムでは、演奏情報のデータ量を効率的に圧縮することが求められている。
【0003】一方、テキスト等のデジタルデータを可逆的(ロスレス)に圧縮する方法としては、文字列(データパターン)が繰り返すことを利用して繰り返し分を圧縮するLZ(Lempel-Zif)法が現在広く使用され、LZ法は一般に使われている可逆式圧縮方法の中で、圧縮率が最も高いと言われている。LZ法は文字列が繰り返すことを利用して圧縮するので、入力データの中の限られた範囲内に出現する同一のデータパターンの数が多く、且つ同一データパターン長が長い場合に圧縮率が高くなるという性質を有する。
【0004】
【発明が解決しようとする課題】しかしながら、SMFではファイル中に出現する同一データパターンの数や長さが必ずしも上記LZ法の条件を満たしていないので、SMFをLZ法により圧縮しても十分な圧縮率を実現することができないという問題点がある。その理由を以下の例で説明すると、先ず、通常の楽曲では1番、2番のように同じようなメロディが繰り返し、したがって、楽譜上でもそれらが同じ音譜で表現されているので、SMFにおいても同様に同一のデータパターンが繰り返して出現しているかのようである。
【0005】しかしながら、楽譜で表すと同一であるメロディであっても、SMFデータでは完全に同一ではないことが多い。図19は一例として2つの似たようなメロディ「1」、「2」のSMFデータを示し、各SMFデータは„タイム、ステータス、ノートナンバ及びベロシティ(強さ)より成る。この場合、楽曲の中の同じようなメロディであっても、繰り返しの単調さを避けたり、メリハリを付ける目的のために„タイムやベロシティ(強さ)が微妙に異なっていることが多く、したがって、LZ法により圧縮しても十分な圧縮率が得られない。
【0006】本発明は上記従来の問題点に鑑み、LZ法により演奏情報のデータ量を効率的に圧縮、また、伸長することができる演奏情報圧縮装置及び演奏情報復号装置を提供することを目的とする。
【0007】
【課題を解決するための手段】本発明は上記目的を達成するために、LZ法により圧縮する前に予め、同一のデータパターンの長さが長く、出現回数が多く且つ近い距離で出現するように、演奏情報を音程と、強さと、長さとその他の情報に分離し、それぞれ独立した領域に配置するようにしている。すなわち本発明によれば、演奏情報を少なくとも音程の情報と、音の強さの情報と、音の長さの情報とその他の情報に分離し、各情報をそれぞれ独立した領域に配置した1次符号を生成する1次符号生成手段と、前記1次符号生成手段により生成された1次符号の各領域の情報をLZ法により圧縮する2次符号生成手段とを有する演奏情報圧縮装置が提供される。
【0008】1次符号生成手段が、各イベント間の相対時間の公約数及び各音符の長さの公約数を算出し、各イベント間の相対時間及び各音符の長さの値をこれらの公約数で除算した後符号化するよう構成されていることは本発明の好ましい態様である。また、1次符号生成手段が、各音符の音程情報をそれ以前に出現した音符の音程の数値を使って一定の関数式に従って算出した数値と実際の音程の数値との残差で表すよう構成されていることは本発明の好ましい態様である。また、1次符号生成手段が、各音符の強さ情報をそれ以前に出現した音符の強さの数値を使って一定の関数式に従って算出した数値と実際の強さの数値との残差で表すよう構成されていることは本発明の好ましい態様である。
【0009】また、1次符号生成手段が、特定の種類のイベントのパラメータ値をそれ以前に出現した同種類のイベントのパラメータ値を使って一定の関数式に従って算出した数値と実際のパラメータ値との残差で表すよう構成されていることは本発明の好ましい態様である。また、1次符号生成手段が、各情報を当該領域においてトラック順に配置することは本発明の好ましい態様である。さらに、1次符号生成手段が、前記各領域をデータの性質が似ている領域同志が近くなるように配置することは本発明の好ましい態様である。
【0010】また、本発明によれば、演奏情報を少なくとも音程の情報と、音の強さの情報と、音の長さの情報と、その他の情報に分離し、前記各情報をそれぞれ独立した領域に配置した1次符号を生成し、この1次符号の各領域の情報をLZ法により圧縮した2次符号を復号する演奏情報復号装置であって、前記2次符号を前記1次符号に復号する2次符号復号手段と、前記2次符号復号手段により復号された1次符号を演奏情報に復号する1次符号復号手段とを有することを特徴とする演奏情報復号装置が提供される。
【0011】
【発明の実施の形態】以下、図面を参照して本発明の実施の形態について説明する。図1は本発明に係る演奏情報圧縮装置の一例を示すブロック図、図2は図1の1次符号生成手段の一例を詳細に示すブロック図、図3は図2のチャネル分離手段により作成されるチャネルマップを示す説明図、図4は図2の解析手段の処理を説明するためのフローチャート、図5は図2の解析手段により作成されるノートテーブルを示す説明図、図6は図2の解析手段により作成されるコントローラテーブルを示す説明図、図7は音符を表現するSMFの„タイムと本実施例のデュレーションの関係を

示す説明図、図8は図2のノート„符号生成手段の処理を説明するためのフローチャート、図9は図2のノート„符号生成手段の処理を説明するた
成手段により生成されるノート„符号を示す説明図、図10は図2のデュレーション符号生成手段の処理を説明するた
めのフローチャート、図11は図2のデュレーション符号生成手段により生成されるデュレーション符号を示す説明図、
図12は図2のノートナンバ符号生成手段により生成されるノートナンバ符号を示す説明図、図13は図2のベロシティ
符号生成手段により生成されるベロシティ符号を示す説明図、図14は図2のコントローラ符号生成手段により生成さ
れるコントローラ符号を示す説明図、図15はSMFの連続イベントブロックを示す説明図、図16は本実施例の連続イ
ベントブロックを示す説明図、図17は図16の連続イベントブロックの効果を示す説明図、図18は図2の符号配置手
段により並べ替えられた1次符号を示す説明図である。

【0012】先ず、入力データ1は一例として図7、図19に示すようなSMFであり、SMFのフォーマットは„タイム、ステー
タス、ノートナンバ及びベロシティにより構成されている。ここで、本明細書では「発音開始イベント」を「ノートオンイベ
ント」、「発音停止イベント」を「ノートオフイベント」と呼ぶ。また「ノートオンイベント」と「ノートオフイベント」を合わせて
「ノートイベント」と呼び、それ以外の「イベント」を「コントローライベント」と呼ぶ。

【0013】図1において、SMFフォーマットの入力データ1は1次符号生成手段2により解析され、少なくとも演奏情報を
音程と、強さと、長さとその他の情報に分離され、各情報がそれぞれ独立した領域に配置した1次符号3が生成され
る。この1次符号3の各領域の符号は2次符号生成手段4によりLZ法で圧縮され、2次符号5が生成される。なお、こ
のように圧縮されたデータは図20～図28に詳しく示す復号装置によりLZ法で復号され、音程と、強さと、長さとその
他の情報に基づいて音符が復元される。

【0014】1次符号生成手段2は例えば図2に詳しく示すように、チャネル分離手段11と、解析手段12と、ノート„符号
生成手段13と、コントローラ„符号生成手段14と、デュレーション符号生成手段15と、ノートナンバ符号生成手段16
と、ベロシティ符号生成手段17と、コントローラ符号生成手段18と符号配置手段19で構成され、この例では1つの音
符を示すノート„符号、コントローラ„符号、デュレーション符号、ノートナンバ符号、ベロシティ符号及びコントローラ符
号の6種類の1次圧縮符号が符号配置手段19により並べ換えられ、1次符号3として2次符号生成手段4に出力さ
れ、2次符号生成手段4により2次圧縮される。

【0015】チェネル分離手段11ではSMF1の1トラックに複数チャネルのイベントが含まれているか否かのチェックを
行ない、もし複数チャネルのイベントが含まれている場合は、1トラックの中に1チャネルのイベントのみが含まれるよ
うに、トラックの分割を行う。そして、図3のようなトラックとチャネル番号の対応を表したチャネルマップを作成し、これ
以降の処理はトラック単位で行う。ここで、SMFのイベントの大半はチャネル情報を含んだものであるが、このようにト
ラック分割とチャネルマップの作成を行うことにより、個々のイベントのチャネル情報を省略することができ、データ量
を削減することができる。

【0016】解析手段12では図4に示す処理を行い、トラック毎に図5に示すようなノートテーブルと図6に示すようなコン
トローラテーブルを作成する。先ず、SMFから順次„タイムとイベントを読み出し(ステップS1)、„タイムからトラックの
先頭を基準としたイベントの時間(以下では単に、イベントの時間と呼ぶ)を計算する(ステップS2)。次にイベントを解
析し、イベントを「ノートオンイベント」、「ノートオフイベント」、「コントローライベント」の3種類に分類する。

【0017】そして、「ノートオンイベント」の場合には図5に示すようなノートテーブルにノートナンバ(音符の音程)とベロ
シティ(音符の強さ)を登録し(ステップS3→S4)、「ノートオフイベント」の場合にはデュレーション(音符の長さ)を計算
してノートテーブルに登録する(ステップS5→S6)。また、「コントローライベント」の場合には図6に示すようなコントロ
ーラテーブルに登録し(ステップS7)、このようにして演奏情報毎にノートテーブルとコントローラテーブルを作成する
(ステップS8→S1)。

【0018】ここで、ノートテーブルは図5に示すようにトラックのノート(音符)イベント情報を時間順に並べたものであり、
コントローラテーブルは図6に示すように、トラックのコントローラ(音符以外)の情報を時間順に並べたものである。ま
た、「ノートオンイベント」の場合にノートナンバとベロシティを書き込む際に、イベントの時間をノートテーブルの所定の
欄に書き込み、また、ノートテーブルの「ノートオフ参照」欄を初期値として「0」にセットする。

【0019】また、イベントがノートオフであれば、ノートテーブルを先頭から走査して、ノートオフイベントの時間よりも早
く、かつノートナンバが同じで、かつノートオフ参照欄が「0」にセットされているノートを選び出し、対応させる。そして対
応するノートオンの時間Tonとノートオフの時間Toffとの差(Toff −Ton)を「デュレーション(音符の長さ)」とし、ノートテ
ーブルの「デュレーション」欄に記録するとともに、「ノートオフ参照」欄を「1」にセットする。

【0020】ここで、「デュレーション」という概念はSMFにはないが、これを用いることによりノートオフイベントを省略でき
るので、データ容量が削減できる。SMFにおいて、1つの音符は図7のように1つのノートオンイベントと1つのノートオ
フイベントの組で表され、また、ノートオフイベントの前の„タイムがデュレーションに相当する。ノートオフイベントのノ
ートナンバは、ノートオンイベントとの対応を取る為に必要であり、デュレーションという概念を使ってノートオンとノート
オフの対応を取っておけば不要である。

【0021】また、ノートオフイベントのベロシティは、MIDIデータを受け取るほとんどの音源が実際にはこの値を使用し
ないので、削除しても問題ない。したがって、ノートオフイベント(3バイト)を省略することにより、場合によっては„タイ
ムのデータ量が増えることもあるが、いずれにしてもノートオフイベント省略の効果の方が大きく、1つの音符につき最
大3バイト分を削減することができる。その結果、1つの楽曲の中に1万個程度の音符が含まれている場合も珍しくな
いので、この場合には最大30Kバイトの削減ができることになり、圧縮効果が大きい。

【0022】イベントがノートオン、ノートオフ以外のイベントであれば、イベントの時間とイベントの内容をコントローラテー
ブルに登録し、このようにしてノートテーブルにはNA個のイベントが登録され、コントローラテーブルにはNB個のイベ
ントが登録される。

【0023】次に、ノート„符号生成手段13とコントローラ„符号生成手段14について説明する。この2つは、同じような
処理内容であるので、以下ではノート„符号生成手段13を例に取って説明する。ノート„符号生成手段13は図8に示
すように、先ず、前述したノートテーブルに登録された各イベントに対し、その時間T[i]と1つ前のイベントの時間T[i−
1]との差„T[i]＝(T[i]−T[i−1])を計算し(ステップS11)、ノートテーブルの所定の欄に書き込む(但し、i＝1～NA、
T[0]＝0)。すなわち、各ノートイベント間の相対時間が求まる。

【0024】ここで、SMFにおいて、„タイムは1拍の何分の1かを基本単位にした可変長符号で表され、値が小さいほど必要なバイト数は少なくて済む。例えば、値が127以下であれば1バイトでよいが、値が128以上16383以下であれば、2バイト必要となる。„タイムの基本単位が細かいほど音楽的な表現力は高いと言えるが、それに従って必要なバイト数も増える。一方、実際に楽曲に使われている„タイムを調べると、基本単位の1刻み(1tick)まで使っていないことが多く、したがって、„T[i] の値が必要以上の容量を使って記録されていることが多い。

【0025】そこで、実際に使用されている時間精度を求める為に、ノートテーブルに登録されている全ての相対時間„T[i] に対する最大公約数„Ts を算出する(ステップS12)。この場合、最大公約数を求めるのが困難な場合は、適当な公約数を最大公約数„Ts とする。次いで、„T[i] をSMFと同様の可変長符号で出力する(ステップS13～S17)。ただし、„T[i] の値を可変長符号にする際に、„T[i] を„Ts で除算した値„Ta[i]を符号化する(ステップS15)。したがって、ノート„符号は図9のように最大公約数„Ts と、NA個の値„Ta[i](i=1 ～NA)により構成される。

【0026】ここで、本演奏情報圧縮装置により圧縮された符号を伸長する際には、„Ta[i]の値を読み取り„Ts をかけ合わせることで元の„T[i] が復元され、したがって、SMFの持つ音楽的な表現力を失うことなくデータ量を削減することができる。例えば„タイムの基本単位を一般的に良く使われる480分の1拍とし、„Ts=10である場合を例にすると、SMFでは1拍の長さ„T=480や1／2拍の長さ„T=240を表現するのに各々2バイト必要である。一方、本発明では„Ts で除算して表現することにより„T=48あるいは„T=24を表現すればよいことになり、各々1バイトの使用で済む。また、1拍あるいは1／2拍に相当する„タイムは使用頻度が高いので、これらが1つの„タイムにつき1バイトずつ削減されれば、楽曲全体で相当の容量を削減することができる。

【0027】コントローラ„符号生成手段14は、処理対象がノートテーブルではなくコントローラテーブルであるという点以外はノート„符号生成手段13と全く同じ処理であり、生成されるコントローラ符号の構成も符号の数がNA個からNB個に変わる以外は、図9に示すノート„符号と同じである。

【0028】デュレーション符号生成手段15はノート„符号生成手段13とほぼ同じであり、図10に従って処理を行う。まず、ノートテーブルに登録された各デュレーションの値の最大公約数Ds を算出する(ステップS21)。この場合、最大公約数を求めるのが困難な場合は、適当な公約数を最大公約数Ds とする。デュレーション符号は図11に示すように最大公約数Ds と、NA個の値Da[i](i=1 ～NA)により構成され、最大公約数Ds に続き、各デュレーションの値D[i]をDs で割った値Da[i]を可変長符号として出力する(ステップS22～S26)。ここで、デュレーションは前述したように、SMFのノートオンイベントとノートオフイベントの間の„タイムに相当するので、ノート„符号生成手段13において説明したのと同様な理由により、SMFに比べデータ量が削減される。

【0029】ノートナンバ符号生成手段16では、ノートテーブルに登録されているノートナンバに対し、以下の処理を施すことによりノートナンバ符号を生成する。ここで、あるノートナンバnum[i]を(1)式のようにそれ以前のS個のノートナンバnum[i - 1]、num[i - 2]、....、num[i - S]を変数とする関数f()と残差„[i]で表す(ただし、num[i - 1]はnum[i]の1つ前のノートナンバ、num[i - 2]はnum[i]の2つ前のノートナンバを表す)。

【0030】ノートナンバ符号は図12に示すように、i≦Sのイベントに対するノートナンバと、i>Sのイベントに対して残差„[i]を時間順に並べたものにより構成される。したがって、圧縮時と伸長時で同じ関数f()を使えば、残差„[i]からnum[i]を復元することができる。

【0031】

【数1】num[i]= f(num[i - 1], num[i - 2],....., num[i - S]) + „[i] …(1)

但し、イベントの数をNAとして、i=(S+1),(S+2),....,NA【0032】ここで、関数f()には種々のものが考えられるが、なるべく同じ値の„[i]が繰り返して出現するようなものを選ぶことにより、2次符号生成手段4で効率の良い圧縮が可能となる。一例として(2)式のような関数を使った場合の効果を説明する。これはS＝1であり、1つ前のノートナンバとの差分を„[i]とすることを意味する。ただしi＝1の場合はノートナンバそのものをノートナンバ符号として出力する。

【0033】

【数2】num[i] = num[i - 1] + „[i] …(2)

但し、イベントの数をNAとして、i=2,3,....,NA【0034】ここで、通常の楽曲においては、コード(和音)のルート音(根音)の平行移動量と同じ音程だけ移動したメロディーラインが存在することが多い。例えば「C」コードの小節で「ド、ド、ミ、ソ、ミ」というメロディーラインがある場合に、ルート音が2度高い「D」コードの小節において「レ、レ、#ファ、ラ、レ」というように、最初のメロディーラインを2度上げたメロディーラインが存在することが多い。

【0035】この各々のメロディーラインをSMFのノートナンバそのもので表すと、「60, 60, 64, 67, 60」、「62, 62, 66, 69, 62」となり、この2つに共通のデータパターンは全くない。しかし前述の„[i]で表現すると、どちらのメロディーラインもその2音目以降は「0, 4, 3, -7」となり同一のパターンとなる。このようにSMFでは全く異なる2つのデータパターンを、本手法により同一のデータパターンに変換することができる。

【0036】LZ法では、同一のデータパターンが多いほど圧縮率が高くなるので、このようなノートナンバの表現方法により圧縮率が高くなることは明らかである。なお(1)式でS＝0とすると、num[i]=„[i]となり、ノートナンバそのものを符号化することになる。また関数f()を何種類か用意しておき、最も適切な関数を選択して符号化するとともに、どの関数を使用したかという情報を合わせて符号化してもよい。

【0037】ベロシティ符号生成手段17もノートナンバ符号生成手段16と同様である。ノートテーブルに登録されたある音符のベロシティvel[i]を(3)式のように、それ以前に出現したT個の音符のベロシティvel[i - 1]、vel[i - 2]、....、vel[i - T]を変数とする関数g()と残差„[i]で表す(ただし、vel[i - 1]はvel[i]の1つ前のベロシティ、vel[i - 2]はvel[i]の2つ前のベロシティ)。

【0038】ベロシティ符号は図13に示すように、i≦Tのイベントに対するベロシティと、i>Tのイベントに対して残差„[i]を時間順に並べたものにより構成される。したがって、圧縮時と伸長時で同じ関数g()を使えば、残差„[i]からvel[i]を復元でき、また、関数g()を適当に選ぶことにより、同じデータパターンの„[i]が繰り返して出現することになり、LZ法を用いた場合の圧縮率を改善することができる。

【0039】

【数3】vel[i]＝g(vel[i－1], vel[i－2],....., vel[i－T]) +„[i] …(3)
但し、イベントの数をNAとして、i=(T+1),(T+2),.....,NA【0040】次にコントローラ符号生成手段18について説明する。コントローラ符号は図14に示すように、図6に示すコントローラテーブルに登録されたイベントの情報を時間順に並べたものである。各コントローラ符号は、イベントの種類を表すフラグFとパラメータ(データバイト)で構成される。パラメータの個数はイベントの種類により異なる。イベントの種類は大きく分けて「通常イベント」と「連続イベント」の2つのタイプがある。フラグFの最上位ビットが「1」、パラメータの最上位ビットが「0」に設定されているので、SMFと同様のランニングステータス表現(前のイベントと同じ種類のイベントの場合にフラグFを省略すること)が可能になっている。
【0041】ここで、SMFではイベントの種類をあらわすのに1バイトのMIDIステータスが使われている。一般的に使用される値は、8n(hex)、9n(hex)、An(hex)、Bn(hex)、Cn(hex)、Dn(hex)、En(hex)、F0(hex)、FF(hex)のいずれかである(ただし、n=0～F(hex)、nはチャネル番号である)。「通常イベント」は、上記MIDIステータスからノートオン8n(hex) とノートオフ9n(hex) を除いたものであるが、本発明では前述したようにチャネル番号を表現する必要が無いので、「通常イベント」のフラグの種類は7種類となる。従ってMIDIステータスに比べフラグFは同じ値になる確率が高く、LZ法を用いた場合の圧縮率が高まる。「通常イベント」の符号は、フラグFの後に、SMFのMIDIステータス1バイトを除いたデータバイトを並べたものである。
【0042】また、SMFでは、特定の種類のイベントが一定数以上連続して出現し、各々のイベントのパラメータ値(データバイト)がほぼ一定の規則で変化する部分が存在することが多い。例えば「ピッチホイールチェンジ」イベントが使われている部分である。このイベントは、音符の音程を微妙に変えて音楽的な表現力を高める為のものであり、その性質上パラメータ値の異なる複数イベントが連続して使われることが多い。このようなイベントを「連続イベント」と呼び、このような部分を「連続イベントブロック」と呼ぶ。
【0043】以下の説明では、「連続イベント」の一例として「ピッチホイールチェンジ」を取りあげるが、これに限定されるものではない。SMFの「連続イベントブロック」の一例を図15に示す。この場合、各イベントのパラメータ値が異なる為、SMFの同一データパターンの長さは、(„タイム・1バイト＋ステータス・1バイト)の計2バイトであり、この程度の長さではLZ法による圧縮の効果はほとんど得られない。
【0044】そこで、コントローラテーブルの中で「ピッチホイールチェンジ」が一定数以上連続して出現し、パラメータ値がほぼ一定の規則で変化する領域に対し、以下の処理を施すことによりコントローラ符号を生成する。先ず、「ピッチホイールチェンジ」の数が一定数に満たないような場合は、前述した「通常イベント」として符号化する。そして、(4)式に示すように、連続イベントブロック内のイベントのパラメータp[i] をそれ以前に出現したU個のイベントのパラメータ値p[i－1] 、p[i－2] 、.....、p[i－U]を変数とする関数h()と残差„[i]で表す(ただし、p[i－1] はp[i] の1つ前のパラメータ値、p[i－2] はp[i] の2つ前のイベントのパラメータ値)。
【0045】連続イベントの符号の構成は図16に示すように、ピッチホイールチェンジが連続していることを意味するフラグFに続き、1番目からU番目までのイベントに対してはパラメータの値そのものである。そして、(U＋1)番目以降のイベントでは、„[i]を時間順に並べたものである。
【0046】
【数4】p[i] = h(p[i－1], p[i－2],....., p[i－U]) +„[i] …(4)
ただし、連続イベントブロックのイベント数をNCとしてi = (U + 1), (U + 2), ..., NC【0047】関数h()には種々のものが考えられるが、なるべく同じ値の„[i]が繰り返して出現するようなものを選ぶことにより、2次符号生成手段4において効率の良い圧縮が可能となる。一例として(5)式のような関数を使った場合の効果を説明する。これはU＝1であり、1つ前のパラメータとの差分をとることを意味する。
【0048】
【数5】
p[i] = p[i－1] + „[i] …(5)
ただし、連続イベントブロックのイベント数をNCとしてi = 2,3, ..., NC【0049】この方法によれば、図15に示す領域は図17のようなコントローラ符号に変換される。この場合、2番目以降のイベントのデータが全て同一の「1」になる為、LZ法による圧縮率が高まる。また、コントローラ符号には„タイムが含まれないので、„タイムがイベント毎に異なる場合でも、LZ法における圧縮率低下の影響が少ない。また場合によっては、(4)式の代わりに、イベントの時間情報も変数に使った(6)式のような関数e()を使っても良い。ただし、t[i]はパラメータを求めるイベントの時間、t[i－1] はその1つ前のイベントの時間である。
【0050】
【数6】p[i] = e(p[i－1],p[i－2],.....,p[i－U],t[i],t[i－1],.....,t[i－U]) +„[i] …(6)
ただし、連続イベントブロックのイベント数をNCとしてi = (U + 1), (U + 2), ..., NC【0051】符号配置手段19では、上記の各符号を図18のような領域に配置して1次符号3を生成する。ヘッダは各符号の開始アドレスや長さといった管理情報と前述したチャネルマップを含んでいる。すでに説明したように各符号はSMFに比べ、同一データの出現回数が多く、同一データパターンの長さも長いという性質を持っているが、さらに同一データパターンが近い距離で出現するように工夫をしている。まず同じ種類の符号内で、同じデータ列が出現する確率が高いので、トラック順に同一種類の符号を配置している。また、ノート„符号とコントローラ„符号とデュレーション符号は、全て時間情報であり、性質の異なるノートナンバ符号やベロシティ符号よりも同じデータ列が出現する確率が高いので、これらの距離が近くなるように配置している。
【0052】次に、図19で示した例に戻って、同一データパターンの長さがどの程度改善されるか具体的に検討する。ここで、各々のメロディは、50個のノートオンイベントと50個のノートオフイベントで構成されており、全ての„タイムは1バイトであるとし、全てのイベントは3バイトであると仮定すると、各々のメロディのノートナンバは前述したように全て同じである。
【0053】SMFにおいて、各々のメロディのデータ量は、(1＋3)×50×2＝400バイトである。各々のメロディの„タイムとベロシティが全て同じならば、同一データパターンの長さは400バイトになる。しかし両メロディ間の全ての„タイムとノートオンのベロシティが異なっているとすると、SMFで同一データパターンの最大長は、ノートオフステータス、ノ

ートナンバ、ベロシティの並びの3バイトである。この程度ではLZ法の圧縮はほとんど効果がない。
【0054】一方、本発明では、„タイム、ノートナンバ、ベロシティを分離して符号化しているので、少なくともノートナンバ符号の中で50バイトの長さの同一データパターンが出現する。また前述したようにSMFのベロシティが全く異なる場合でも、ベロシティ符号の中では同一データパターンが出現することが多い。従ってLZ法による圧縮率は明らかに改善される。以上の説明から分かるように1次符号3は、SMFの持つ音楽的な情報量を全く落とすことなくデータ量が削減されていると同時に、SMFに比べ同一のデータパターンの長さが長く、出現回数が多く、しかもそれらが近い距離で出現するようの性質を持っているので、2次符号生成手段4において効果的な圧縮を行うことができる。また、この1次符号3そのものも、かなり圧縮されたデータ量となっているので、この1次符号3を直接出力するようにしてもよい。
【0055】2次符号生成手段4においては、1次符号生成手段2の出力3に対して、LZ法による圧縮を行う。LZ法は、gzip、LHAといった圧縮プログラムで広く使われている手法である。これは入力データの中から、同一のデータパターンを捜し、もし存在すれば、(過去に出現したパターンへの距離、パターンの長さ)という情報に置き換えて表現することでデータ量を削減する。例えば、"ABCDEABCDEF"というデータを、"ABCDE"が繰り返しであるので、"ABCDE(5, 5)F"という情報に置き換える。なお、圧縮符号(5, 5)は5文字戻って5文字コピーすることを表す。
【0056】処理の概要は次のようになる。2次符号5の生成は、処理位置を1次符号3の先頭から順次移動させて行う。処理位置のデータパターンが、それ以前の一定の範囲内のデータパターンと一致する場合は、処理位置からそのデータパターンまでの距離と、一致したデータパターンの長さを2次符号5として出力し、処理位置を2つ目のデータパターンの終わりに移動させ、処理を続ける。処理位置のデータパターンが、それ以前の一定の範囲内のデータパターンと一致しなければ、1次符号3をコピーして2次符号5として出力する。
【0057】以上の説明から明らかなように、2つの同一のデータ領域が大きいほど、圧縮率は高くなる。また同一のデータ領域の距離は一定範囲内である必要がある。前述したように楽曲の中では、同じようなメロディーが繰り返し使われるが、SMFのままでそれらのデータを比較すると、完全に同一のデータ列の繰り返しであることは少なく、むしろノートナンバは同じであるがベロシティは異なるといったように、どこか一部分異なっていることが多い。
【0058】一方，本発明では、性質の同じデータを独立した領域にまとめると同時に、各領域で同一のデータがなるべく多く出現するような処理を行ない、さらに性質の近い領域どうしをなるべく近くに配置することにより、LZ法の圧縮率が高まるので、最終的な2次符号5は十分容量の小さなものになる。なお、以上詳述したフォーマット並びに処理手順は一例であり、その主旨を逸脱しない範囲において種々の変更を加えることができる。また、演奏情報としてSMFを例に取ったが、SMFに限らずこれに類似の演奏情報に対して本発明を適用してデータ容量を効率よく削減することができる。
【0059】次に、図20～図28を参照して上記の1次符号3又は2次符号5を復号するための演奏情報復号装置について説明する。図20は演奏情報復号装置を示すブロック図、図21は図20の2次符号復号手段の処理を説明するためのフローチャート、図23は図22のトラック復号処理を詳しく説明するためのフローチャート、図22は図20の1次符号復号手段の処理を説明するためのフローチャート、図24は図23のノートイベント復号処理を詳しく説明するためのフローチャート、図25は図24のノートイベント復号処理により復元されたノートオンイベントを示す説明図、図26は図24のノートイベント復号処理により復元されたノートオフキューを示す説明図、図27は図23のコントローライベント復号処理を詳しく説明するためのフローチャート、図28は図27の処理により復元されたコントローライベントを示す説明図である。
【0060】図20では圧縮処理とは逆に、LZ法で圧縮された入力データ21が2次符号復号手段23により音程と、音の強さと、音の長さとその他の情報に分離された1次符号3に復号され、次いで1次符号復号手段24により元の音符(出力データ25)に復元される。制御手段26はスイッチ22により、入力データ21が図1に示す2次符号5である場合に2次符号復号処理に続き1次符号復号処理を行うように制御し、入力データ21が図1に示す1次符号3である場合に1次符号復号処理のみを行うように制御する。
【0061】ここで、2次符号5であるか又は1次符号3であるかの判定は、キーボード、マウス、ディスプレイ等の図示しない入出力装置を使用してオペレータが指定してもよいし、圧縮された情報に対して符号化方法の種類を示す情報を符号化時に付加し、復号時にこの情報を自動的に判別するようにしてもよい。
【0062】次に、図21を参照して2次符号復号手段23の復号処理を説明する。入力データ11(2次符号5)を先頭から読み込み(ステップS101)、次いで圧縮データの部分であるかすなわちABCDE(5, 5)の「ABCDE」の部分(＝非圧縮データ)であるか又は「(5, 5)」の部分(＝圧縮データ)であるかを判定する(ステップS102)。
【0063】そして、圧縮データの部分である場合には過去に出現した同一パターンを参照してそれをコピーして出力し(ステップS103)、他方、非圧縮データの部分である場合にはそのまま出力する(ステップS104)。以下、入力データ11を全て復号するまでこの処理を繰り返すと(ステップS105→S101)、図18に示すような配置の1次符号3が復元される。
【0064】次に、図22及び図18に示す1次符号3を参照して1次符号復号手段24の復号処理を説明する。先ず、1次符号3のヘッダを読み込む(ステップS111)。ヘッダには総トラック数N、ノート„符号からコントロール符号までの各符号領域の先頭アドレス、チャネルマップ、時間分解能等の情報が符号化の際に記録されているので、このヘッダ情報に基づいてSMFのヘッダを作成して出力する(ステップS112)。
【0065】次にトラック番号iを「1」にセットし(ステップS113)、図23に詳しく示すトラック復号処理を行う(ステップS114)。次いでトラック番号iが総トラック数Nより小さいか否かをチェックし(ステップS115)、もし小さければトラック番号iを1つインクリメントし(ステップS116)、ステップS114に戻ってトラック復号処理を繰り返す。そして、ステップS115においてトラック番号iが総トラック数Nより小さくない場合にこの1次符号復号処理を終了する。
【0066】図23に詳しく示すトラック復号処理では、先ず、処理で使用する変数を初期化する(ステップS121)。具体的は、処理中のノートイベントの番号を示す変数jを「1」にセットし、処理中のコントローライベントの番号を示す変数kを「1」にセットし、ノート終了フラグとコントローラ終了フラグをクリアする。ここで、ノート終了フラグは処理トラックの全てのノートイベントの復号が終了したことを示し、コントローラ終了フラグは処理トラックの全てのコントローライベントの復号が終了したことを示す。

【0067】次に処理トラック番号iのノート„ 符号の最大公約数„ Tsnと、コントローラ„ 符号の最大公約数„ Tscとデュレーション符号の最大公約数Ds を読み出す(ステップS122)。そして、j番目のノート„ 符号„ Tan[j] とk番目のコントローラ„ 符号„ Tac[k] を読み出し、(7)式のように各々最大公約数„ Tsn、„ Tscを乗じて„ Tn[j]、„ Tc[k]を算出する(ステップS123)。

„ Tn[j]＝„ Tan[j] ×„ Tsn„ Tc[k]＝„ Tac[k] ×„ Tsc …(7)

【0068】さらに、(8)式のようにトラックの先頭を基準とした時刻Tn[j]、Tc[k]に変換する(ステップS124)。

Tn[j]＝Tn[j-1]＋„ Tn[j]Tc[k]＝Tc[k-1]＋„ Tc[k]ただし、Tn[0]＝Tc[0]＝0 …(8)

なお、ステップS123、S124では、ノート終了フラグがセットされている場合には„ Tn[j]、Tn[j]の算出は行わず、また、コントローラ終了フラグがセットされている場合には„ Tc[k]、Tc[k]の算出は行わない。

【0069】次に、出力すべきノートオフイベントの有無をチェックし(ステップS125)、出力すべきデータが有る場合にはSMFとしてノートオフイベントを出力する(ステップS126)。なお、ステップS125、S126については後述(図24のステップS144)する。次に、復号処理の選択を行う。先ず、コントローラ終了フラグをチェックし(ステップS127)、セットされている場合には図24に詳しく示すノートイベント復号処理を行う(ステップS128)。

【0070】コントローラ終了フラグがセットされていない場合にはノート終了フラグをチェックし(ステップS129)、セットされている場合には図27に詳しく示すコントローライベント復号処理を行う(ステップS130)。2つのフラグが共にセットされていない場合にはTn[j]とTc[k]を比較し(ステップS131)、Tn[j]が小さければノートイベント復号処理(ステップS128)を、そうでなければコントローライベント復号処理(ステップS130)を行う。

【0071】ノートイベント復号処理の後には、処理トラックNの全てのノートイベントを処理したか否かをチェックし(ステップS132)、処理が終了している場合にはノート終了フラグをセットし(ステップS133)、ステップS138に進み、そうでなければ変数jを1つインクリメントし(ステップS134)、ステップS123に戻る。また、コントローライベント復号処理の後には、処理トラックNの全てのコントローライベントを処理したか否かをチェックし(ステップS135)、処理が終了している場合にはコントローラ終了フラグをセットし(ステップS136)ステップS138に進み、そうでなければ変数kを1つインクリメントし(ステップS137)、ステップS123に戻る。

【0072】ステップS138ではノート終了フラグとコントローラ終了フラグの両方がセットされているか否かをチェックし、両方がセットされている場合にはこのトラック復号処理を終了し、そうでなければステップS123に戻ってこのトラック復号処理を繰り返す。

【0073】図24に詳しく示すノートイベント復号処理では、先ず、j番目のノートナンバ符号„[j] を読み取り、圧縮処理において使用した関数f()を用いて(9)式に従ってノートナンバnum[j]を算出する(ステップS141)。

【0074】

【数7】

num[j]＝f(num[j-1],num[j-2],.....,num[j-S]) + „[j] (j＞S) num[j]＝„[j] (j≦S) ただし、S は関数f()の変数の個数 …(9)

【0075】同様に、j番目のベロシティ符号„[j] を読み取り、圧縮処理において使用した関数g()を用いて(10)式に従ってベロシティvel[j]を算出する(ステップS142)。

【0076】

【数8】

vel[j]＝g(vel[j-1],vel[j-2],.....,vel[j-T]) + „[j] (j＞T) vel[j]＝„[j] (j≦T) ただし、T は関数g()の変数の個数 …(10)

【0077】次いでTn[j]、num[j]、vel[j]を用いて図25に示すようなノートオンイベントを出力する(ステップS143)。なお、SMTの„ タイム„ Tは、Tn[j]の直前に出力したイベントの時刻Tb を使って式(11)に従って求め、出力する。

„ T＝Tn[j]－Tb …(11)

図25に示すノートオンイベントにおけるステータスバイトの上位4ビットはノートオン「9(hex)」を表し、下位4ビットはチャネルマップから得られる番号が続く。このステータスバイトの後にはノートナンバとベロシティの各バイトが続く。

【0078】次にノートオフイベントの登録を行う(ステップS144)。具体的にはデュレーション符号Da[j]を読み取って、(12)式に従いノートオフイベントの時刻Toff を算出し、この時刻Toff とノートナンバnum[j]を図26に示すようなノートオフキューに登録する。このノートオフキューでは、使用されているエントリの数を保持するとともに、ノートオフ時刻Toff が先頭から小さい順に並ぶように管理される。

Toff[j]＝Da[j]×Ds ＋Tn[j] …(12)

【0079】前述した図23のステップS125においては、Tn[j]とTc[k]の内の値が小さいほうTm をノートオフキューの先頭のToff[n](n＝1～エントリ総数N)から順に比較する。Toff[n]＜Tm であるエントリがあればステップS126に進み、ノートオフイベントを出力する。ステップS126では、前述したノートオフイベントをSMFとして出力する。

【0080】次に図27を参照してコントローライベント復号処理を詳しく説明する。この処理では図28に示すように„ タイム、ステータス及びパラメータより成るコントローライベントが復元され、先ず、Tc[k]と直前に出力したイベントの時刻Tb を使って(13)式に従ってSMTの„ タイム„ Tを求め、出力する(ステップS151)。

„ T＝Tc[k]－Tb …(13)

【0081】次にコントローラ符号領域からイベントの種類を表すイベントフラグF[k] を読み取り、F[k] が「通常イベント」であるか、「連続イベント」であるか又は「ランニングステータス」であるかを判定する(ステップS152)。ここで、連続イベントブロック内では、図示省略16に示すように、2番目以降のイベントはイベントフラグが省略された「ランニングステータス」状態で記録されている。

【0082】F[k] が「通常イベント」である場合には、処理イベントの連続イベントブロック内における順番を示す変数mを「0」にリセットし(ステップS153)、次いでチャネルマップを参照してSMFのステータスバイトを作成して出力する(ステップS154)。さらにイベントの種類に応じて必要なバイト数をコントローラ符号領域から読み出し、この読み出した値がSMFのパラメータ(データバイト)であるのでこれを出力する(ステップS155)。

【0083】F[k] が「連続イベント」である場合には、連続イベントブロック内における順番を示す変数mを「1」にセットし(ステップS156)、次いでチャネルマップを参照してSMFのステータスバイトを作成して出力する(ステップS157)。なお、m≧2の場合のステータスバイトはmが「1」の場合のステータスバイトを利用する。そして、この「連続イベント」

の場合には、パラメータ符号„[m] を読み出し、圧縮処理と同じ関数h()を使い、(14)式に従ってSMFのパラメータp[m] を作成し、出力する(ステップS158)。

【0084】

【数9】

$$p[m] = h(p[m-1], p[m-2], \ldots\ldots, p[m-U]) + „[m] \quad (m > U)$$

$$p[m] = „[m] \quad (m \leqq U)$$

ただし、Uは関数h()の変数の個数 …(14)

【0085】F[k] が「ランニングステータス」である場合には、変数mの値をチェックし(ステップS159)、mが「0」より大きければmを1つインクリメントし(ステップS160)、「連続イベント」側のステップS157に進む。他方、mが「0」であれば「通常イベント」側のステップS154に進む。

【0086】

【発明の効果】以上説明したように本発明によれば、LZ法により圧縮する前に予め、同一のデータパターンの長さが長く、出現回数が多く且つ近い距離で出現するように、演奏情報を音程と、強さと、長さとその他の情報に分離し、各情報をそれぞれ独立した領域に配置した1次符号を生成し、この1次符号をLZ法により圧縮するので、演奏情報のデータ量を効率的に圧縮することができる。

【0087】また、1次符号として演奏情報を音符の音程領域と、音符の強さ領域と、音符の長さ領域とその他の領域の少なくとも4つの領域に分離して符号化するので、元の演奏情報のもつ演奏品位を全く失うことなく、従来に比べ大幅にデータ容量が削減でき、したがって、データを保存するのに小容量の記録媒体を用いることができコストを削減することができ、また、通信回線を介してデータを伝送する場合にもコストを削減することができるとともに、伝送時間を削減することができる。また、大量の演奏情報を扱う通信カラオケや音楽データベースで特に効果が大きい。

分野

【発明の属する技術分野】本発明は、演奏情報のデータ量を圧縮する演奏情報圧縮装置及び演奏情報の圧縮データを復号する演奏情報復号装置に関する。

技術

【従来の技術】一般に、MIDI(Musical Instrument Digital Interface)データを保存する方式として、スタンダードMIDIファイル(以下、SMFという。)が広く用いられ、このSMFでは図19に示すように個々の演奏情報がデルタ(„)タイムとイベントの2つの要素により構成されている。„タイムは、隣合ったイベント間の相対時間を表し、イベントはノートオン又はノートオフのステータス、音程(ノートナンバ)や音の強さ(ベロシティ)などの種々の演奏情報を含む。ここで、演奏情報とは音符により示される音程、音の長さの他に、音の強さ、楽曲の演奏上の拍子、テンポなどの情報、さらに音源の種類、リセットコントロールの情報などを含むものを指すものとする。また、このSMFでは各演奏情報が時間順に並んでトラックを構成している。ここで、SMF方式は記憶容量や伝送路の効率的利用という点ではあまり優れたものではないので、通信カラオケや音楽データベースのように多数の楽曲データを記録、伝送するシステムでは、演奏情報のデータ量を効率的に圧縮することが求められている。

【0003】一方、テキスト等のデジタルデータを可逆的(ロスレス)に圧縮する方法としては、文字列(データパターン)が繰り返すことを利用して繰り返し分を圧縮するLZ(Lempel-Zif)法が現在広く使用され、LZ法は一般に使われている可逆式圧縮方法の中で、圧縮率が最も高いと言われている。LZ法は文字列が繰り返すことを利用して圧縮するので、入力データの中の限られた範囲内に出現する同一のデータパターンの数が多く、且つ同一データパターン長が長い場合に圧縮率が高くなるという性質を有する。

効果

【発明の効果】以上説明したように本発明によれば、LZ法により圧縮する前に予め、同一のデータパターンの長さが長く、出現回数が多く且つ近い距離で出現するように、演奏情報を音程と、強さと、長さとその他の情報に分離し、各情報をそれぞれ独立した領域に配置した1次符号を生成し、この1次符号をLZ法により圧縮するので、演奏情報のデータ量を効率的に圧縮することができる。

【0087】また、1次符号として演奏情報を音符の音程領域と、音符の強さ領域と、音符の長さ領域とその他の領域の少なくとも4つの領域に分離して符号化するので、元の演奏情報のもつ演奏品位を全く失うことなく、従来に比べ大幅にデータ容量が削減でき、したがって、データを保存するのに小容量の記録媒体を用いることができコストを削減することができ、また、通信回線を介してデータを伝送する場合にもコストを削減することができるとともに、伝送時間を削減することができる。また、大量の演奏情報を扱う通信カラオケや音楽データベースで特に効果が大きい。

課題

【発明が解決しようとする課題】しかしながら、SMFではファイル中に出現する同一データパターンの数や長さが必ずしも上記LZ法の条件を満たしていないので、SMFをLZ法により圧縮しても十分な圧縮率を実現することができないという問題点がある。その理由を以下の例で説明すると、先ず、通常の楽曲では1番、2番のように同じようなメロディが繰り返し、したがって、楽譜上でもそれらが同じ音譜で表現されているので、SMFにおいても同様に同一のデータパターンが繰り返して出現しているかのようである。

【0005】しかしながら、楽譜で表すと同一であるメロディであっても、SMFデータでは完全に同一ではないことが多い。図19は一例として2つの似たようなメロディ「1」、「2」のSMFデータを示し、各SMFデータは、タイム、ステータス、ノートナンバ及びベロシティ(強さ)より成る。この場合、楽曲の中の同じようなメロディであっても、繰り返しの単調さを避けたり、メリハリを付ける目的のために、タイムやベロシティ(強さ)が微妙に異なっていることが多く、したがって、LZ法により圧縮しても十分な圧縮率が得られない。

【0006】本発明は上記従来の問題点に鑑み、LZ法により演奏情報のデータ量を効率的に圧縮、また、伸長することができる演奏情報圧縮装置及び演奏情報復号装置を提供することを目的とする。

手段

【課題を解決するための手段】本発明は上記目的を達成するために、LZ法により圧縮する前に予め、同一のデータパターンの長さが長く、出現回数が多く且つ近い距離で出現するように、演奏情報を音程と、強さと、長さとその他の情報に分離し、それぞれ独立した領域に配置するようにしている。すなわち本発明によれば、演奏情報を少なくとも音程の情報と、音の強さの情報と、音の長さの情報とその他の情報に分離し、各情報をそれぞれ独立した領域に配置した1次符号を生成する1次符号生成手段と、前記1次符号生成手段により生成された1次符号の各領域の情報をLZ法により圧縮する2次符号生成手段とを有する演奏情報圧縮装置が提供される。

【0008】1次符号生成手段が、各イベント間の相対時間の公約数及び各音符の長さの公約数を算出し、各イベント間の相対時間及び各音符の長さの値をこれらの公約数で除算した後符号化するよう構成されていることは本発明の好ましい態様である。また、1次符号生成手段が、各音符の音程情報をそれ以前に出現した音符の音程の数値を使って一定の関数式に従って算出した数値と実際の音程の数値との残差で表すよう構成されていることは本発明の好ましい態様である。また、1次符号生成手段が、各音符の強さ情報をそれ以前に出現した音符の強さの数値を使って一定の関数式に従って算出した数値と実際の強さの数値との残差で表すよう構成されていることは本発明の好ましい態様である。

【0009】また、1次符号生成手段が、特定の種類のイベントのパラメータ値をそれ以前に出現した同種類のイベントのパラメータ値を使って一定の関数式に従って算出した数値と実際のパラメータ値との残差で表すよう構成されていることは本発明の好ましい態様である。また、1次符号生成手段が、各情報を当該領域においてトラック順に配置することは本発明の好ましい態様である。さらに、1次符号生成手段が、前記各領域をデータの性質が似ている領域同志が近くなるように配置することは本発明の好ましい態様である。

【0010】また、本発明によれば、演奏情報を少なくとも音程の情報と、音の強さの情報と、音の長さの情報と、その他の情報に分離し、前記各情報をそれぞれ独立した領域に配置した1次符号を生成し、この1次符号の各領域の情報をLZ法により圧縮した2次符号を復号する演奏情報復号装置であって、前記2次符号を前記1次符号に復号する2次符号復号手段と、前記2次符号復号手段により復号された1次符号を演奏情報に復号する1次符号復号手段とを有することを特徴とする演奏情報復号装置が提供される。

【0011】

【発明の実施の形態】以下、図面を参照して本発明の実施の形態について説明する。図1は本発明に係る演奏情報圧縮装置の一例を示すブロック図、図2は図1の1次符号生成手段の一例を詳細に示すブロック図、図3は図2のチャネル分離手段により作成されるチャネルマップを示す説明図、図4は図2の解析手段の処理を説明するためのフローチャート、図5は図2の解析手段により作成されるノートテーブルを示す説明図、図6は図2の解析手段により作成されるコントローラテーブルを示す説明図、図7は音符を表現するSMFの„タイムと本実施例のデュレーションの関係を示す説明図、図8は図2のノート„符号生成手段の処理を説明するためのフローチャート、図9は図2のノート„符号生成手段により生成されるノート„符号を示す説明図、図10は図2のデュレーション符号生成手段の処理を説明するためのフローチャート、図11は図2のデュレーション符号生成手段により生成されるデュレーション符号を示す説明図、図12は図2のノートナンバ符号生成手段により生成されるノートナンバ符号を示す説明図、図13は図2のベロシティ符号生成手段により生成されるベロシティ符号を示す説明図、図14は図2のコントローラ符号生成手段により生成されるコントローラ符号を示す説明図、図15はSMFの連続イベントブロックを示す説明図、図16は本実施例の連続イベントブロックの効果を示す説明図、図18は図2の符号配置手段により並べ替えられた1次符号を示す説明図である。

【0012】先ず、入力データ1は一例として図7、図19に示すようなSMFであり、SMFのフォーマットは„タイム、ステータス、ノートナンバ及びベロシティにより構成されている。ここで、本明細書では「発音開始イベント」を「ノートオンイベント」、「発音停止イベント」を「ノートオフイベント」と呼ぶ。また「ノートオンイベント」と「ノートオフイベント」を合わせて「ノートイベント」と呼び、それ以外の「イベント」を「コントローライベント」と呼ぶ。

【0013】図1において、SMFフォーマットの入力データ1は1次符号生成手段2により解析され、少なくとも演奏情報を音程と、強さと、長さとその他の情報に分離され、各情報がそれぞれ独立した領域に配置した1次符号3が生成される。この1次符号3の各領域の符号は2次符号生成手段4によりLZ法で圧縮され、2次符号5が生成される。なお、このように圧縮されたデータは図20～図28に詳しく示す復号装置によりLZ法で復号され、音程と、強さと、長さとその他の情報に基づいて音符が復元される。

【0014】1次符号生成手段2は例えば図2に詳しく示すように、チャネル分離手段11と、解析手段12と、ノート„符号生成手段13と、コントローラ„符号生成手段14と、デュレーション符号生成手段15と、ノートナンバ符号生成手段16と、ベロシティ符号生成手段17と、コントローラ符号生成手段18と符号配置手段19で構成され、この例では1つの音符を示すノート„符号、コントローラ„符号、デュレーション符号、ノートナンバ符号、ベロシティ符号及びコントローラ符号の6種類の1次圧縮符号が符号配置手段19により並べ換えられ、1次符号3として2次符号生成手段4に出力され、2次符号生成手段4により2次圧縮される。

【0015】チェネル分離手段11ではSMF1の1トラックに複数チャネルのイベントが含まれているか否かのチェックを行ない、もし複数チャネルのイベントが含まれている場合は、1トラックの中に1チャネルのイベントのみが含まれるように、トラックの分割を行う。そして、図3のようなトラックとチャネル番号の対応を表したチャネルマップを作成し、これ以降の処理はトラック単位で行う。ここで、SMFのイベントの大半はチャネル情報を含んだものであるが、このようにトラック分割とチャネルマップの作成を行うことにより、個々のイベントのチャネル情報を省略することができ、データ量を削減することができる。

【0016】解析手段12では図4に示す処理を行い、トラック毎に図5に示すようなノートテーブルと図6に示すようなコントローラテーブルを作成する。先ず、SMFから順次„タイムとイベントを読み出し(ステップS1)、„タイムからトラックの先頭を基準としたイベントの時間(以下では単に、イベントの時間と呼ぶ)を計算する(ステップS2)。次にイベントを解析し、イベントを「ノートオンイベント」、「ノートオフイベント」、「コントローライベント」の3種類に分類する。

【0017】そして、「ノートオンイベント」の場合には図5に示すようなノートテーブルにノートナンバ(音符の音程)とベロシティ(音符の強さ)を登録し(ステップS3→S4)、「ノートオフイベント」の場合にはデュレーション(音符の長さ)を計算してノートテーブルに登録する(ステップS5→S6)。また、「コントローライベント」の場合には図6に示すようなコントローラテーブルに登録し(ステップS7)、このようにして演奏情報毎にノートテーブルとコントローラテーブルを作成する(ステップS8→S1)。

【0018】ここで、ノートテーブルは図5に示すようにトラックのノート(音符)イベント情報を時間順に並べたものであり、コントローラテーブルは図6に示すように、トラックのコントローラ(音符以外)の情報を時間順に並べたものである。また、「ノートオンイベント」の場合にノートナンバとベロシティを書き込む際に、イベントの時間をノートテーブルの所定の欄に書き込み、また、ノートテーブルの「ノートオフ参照」欄を初期値として「0」にセットする。

【0019】また、イベントがノートオフであれば、ノートテーブルを先頭から走査して、ノートオフイベントの時間よりも早く、かつノートナンバが同じで、かつノートオフ参照欄が「0」にセットされているノートを選び出し、対応させる。そして対応するノートオンの時間Tonとノートオフの時間Toffとの差(Toff −Ton)を「デュレーション(音符の長さ)」とし、ノートテーブルの「デュレーション」欄に記録するとともに、「ノートオフ参照」欄を「1」にセットする。

【0020】ここで、「デュレーション」という概念はSMFにはないが、これを用いることによりノートオフイベントを省略できるので、データ容量が削減できる。SMFにおいて、1つの音符は図7のように1つのノートオンイベントと1つのノートオフイベントの組で表され、また、ノートオフイベントの前の„タイムがデュレーションに相当する。ノートオフイベントのノートナンバは、ノートオンイベントとの対応を取る為に必要であり、デュレーションという概念を使ってノートオンとノートオフの対応を取っておけば不要である。

【0021】また、ノートオフイベントのベロシティは、MIDIデータを受け取るほとんどの音源が実際にはこの値を使用しないので、削除しても問題ない。したがって、ノートオフイベント(3バイト)を省略することにより、場合によっては„タイムのデータ量が増えることもあるが、いずれにしてもノートオフイベント省略の効果の方が大きく、1つの音符につき最大3バイト分を削減することができる。その結果、1つの楽曲の中に1万個程度の音符が含まれている場合も珍しくないので、この場合には最大30Kバイトの削減ができることになり、圧縮効果が大きい。

【0022】イベントがノートオン、ノートオフ以外のイベントであれば、イベントの時間とイベントの内容をコントローラテーブルに登録し、このようにしてノートテーブルにはNA個のイベントが登録され、コントローラテーブルにはNB個のイベントが登録される。

【0023】次に、ノート„符号生成手段13とコントローラ„符号生成手段14について説明する。この2つは、同じような処理内容であるので、以下ではノート„符号生成手段13を例に取って説明する。ノート„符号生成手段13は図8に示すように、先ず、前述したノートテーブルに登録された各イベントに対し、その時間T[i] と1つ前のイベントの時間T[i-1] との差„T[i] =(T[i] −T[i-1])を計算し(ステップS11)、ノートテーブルの所定の欄に書き込む(但し、i=1〜NA, T[0] =0)。すなわち、各ノートイベント間の相対時間が求まる。

【0024】ここで、SMFにおいて、„タイムは1拍の何分の1かを基本単位にした可変長符号で表され、値が小さいほど必要なバイト数は少なくて済む。例えば、値が127以下であれば1バイトでよいが、値が128以上16383以下であれば、2バイト必要となる。„タイムの基本単位が細かいほど音楽的な表現力は高いと言えるが、それに従って必要なバイト数も増える。一方、実際に楽曲に使われている„タイムを調べると、基本単位の1刻み(1tick)まで使っていないことが多く、したがって、„T[i] の値が必要以上の容量を使って記録されていることが多い。

【0025】そこで、実際に使用されている時間精度を求める為に、ノートテーブルに登録されている全ての相対時間„T[i] に対する最大公約数„Ts を算出する(ステップS12)。この場合、最大公約数を求めるのが困難な場合は、適当な公約数を最大公約数„Ts とする。次いで、„T[i] をSMFと同様の可変長符号で出力する(ステップS13〜S17)。ただし„T[i] の値を可変長符号にする際に、„T[i] を„Ts で除算した値„Ta[i]を符号化する(ステップS15)。したがって、ノート„符号は図9のように最大公約数„Ts と、NA個の値„Ta[i](i=1 〜NA)により構成される。

【0026】ここで、本演奏情報圧縮装置により圧縮された符号を伸長する際には、„Ta[i]の値を読み取り„Ts をかけ合わせることで元の„T[i] が復元され、したがって、SMFの持つ音楽的な表現力を失うことなくデータ量を削減することができる。例えば„タイムの基本単位を一般的に良く使われる480分の1拍とし、„Ts=10である場合を例にすると、SMFでは1拍の長さ„T=480や1／2拍の長さ„T=240を表現するのに各々2バイト必要である。一方、本発明では„Ts で除算して表現することにより„T=48あるいは„T=24を表現すればよいことになり、各々1バイトの使用で済む。また、1拍あるいは1／2拍に相当する„タイムは使用頻度が高いので、これらが1つの„タイムにつき1バイトずつ削減されれば、楽曲全体で相当の容量を削減することができる。

【0027】コントローラ„符号生成手段14は、処理対象がノートテーブルではなくコントローラテーブルであるという点以外はノート„符号生成手段13と全く同じ処理であり、生成されるコントローラ符号の構成も符号の数がNA個からNB個に変わる以外は、図9に示すノート„符号と同じである。

【0028】デュレーション符号生成手段15はノート„符号生成手段13とほぼ同じであり、図10に従って処理を行う。まず、ノートテーブルに登録された各デュレーションの値の最大公約数Ds を算出する(ステップS21)。この場合、最大公約数を求めるのが困難な場合は、適当な公約数を最大公約数Ds とする。デュレーション符号は図11に示すように最大公約数Ds と、NA個の値Da[i](i=1 〜NA)により構成され、最大公約数Ds に続き、各デュレーションの値D[i]をDs で割った値Da[i]を可変長符号として出力する(ステップS22〜S26)。ここで、デュレーションは前述したように、SMFのノートオンイベントとノートオフイベントの間の„タイムに相当するので、ノート„符号生成手段13において説明したのと同様な理由により、SMFに比べデータ量が削減される。

【0029】ノートナンバ符号生成手段16では、ノートテーブルに登録されているノートナンバに対し、以下の処理を施すことによりノートナンバ符号を生成する。ここで、あるノートナンバnum[i]を(1)式のようにそれ以前のS個のノートナンバnum[i − 1]、num[i − 2]、....、num[i − S]を変数とする関数f()と残差„[i]で表す(ただし、num[i − 1]はnum[i]の1つ前のノートナンバ、num[i − 2]はnum[i]の2つ前のノートナンバを表す)。

【0030】ノートナンバ符号は図12に示すように、i≦Sのイベントに対するノートナンバと、i>Sのイベントに対して残差„[i]を時間順に並べたものにより構成される。したがって、圧縮時と伸長時で同じ関数f()を使えば、残差„[i]からnum

[i]を復元することができる。
【0031】
【数1】num[i]＝ f(num[i − 1], num[i − 2],....., num[i − S]) + „[i] …(1)
但し、イベントの数をNAとして、i=(S+1),(S+2),.....,NA【0032】ここで、関数f()には種々のものが考えられるが、なるべく同じ値の„[i]が繰り返して出現するようなものを選ぶことにより、2次符号生成手段4で効率の良い圧縮が可能となる。一例として(2)式のような関数を使った場合の効果を説明する。これはS＝1であり、1つ前のノートナンバとの差分を„[i]とすることを意味する。ただしi＝1の場合はノートナンバそのものをノートナンバ符号として出力する。
【0033】
【数2】num[i] = num[i − 1] + „[i] …(2)
但し、イベントの数をNAとして、i=2,3,.....,NA【0034】ここで、通常の楽曲においては、コード(和音)のルート音(根音)の平行移動量と同じ音程だけ移動したメロディーラインが存在することが多い。例えば「C」コードの小節で「ド、ド、ミ、ソ、ミ」というメロディーラインがある場合に、ルート音が2度高い「D」コードの小節において「レ、レ、#ファ、ラ、レ」というように、最初のメロディーラインを2度上げたメロディーラインが存在することが多い。
【0035】この各々のメロディーラインをSMFのノートナンバそのもので表すと、「60, 60, 64, 67, 60」、「62, 62, 66, 69, 62」となり、この2つに共通のデータパターンは全くない。しかし前述の„[i]で表現すると、どちらのメロディーラインもその2音目以降は「0, 4, 3, −7」となり同一のパターンとなる。このようにSMFでは全く異なる2つのデータパターンを、本手法により同一のデータパターンに変換することができる。
【0036】LZ法では、同一のデータパターンが多いほど圧縮率が高くなるので、このようなノートナンバの表現方法により圧縮率が高くなることは明らかである。なお(1)式でS＝0とすると、num[i]＝„[i]となり、ノートナンバそのものを符号化することになる。また関数f()を何種類か用意しておき、最も適切な関数を選択して符号化するとともに、どの関数を使用したかという情報を合わせて符号化してもよい。
【0037】ベロシティ符号生成手段17もノートナンバ符号生成手段16と同様である。ノートテーブルに登録されたある音符のベロシティvel[i]を(3)式のように、それ以前に出現したT個の音符のベロシティvel[i − 1]、vel[i − 2]、....、vel[i − T]を変数とする関数g()と残差„[i]で表す(ただし、vel[i − 1]はvel[i]の1つ前のベロシティ、vel[i − 2]はvel[i]の2つ前のベロシティ)。
【0038】ベロシティ符号は図13に示すように、i≦Tのイベントに対するベロシティと、i>Tのイベントに対して残差„[i]を時間順に並べたものにより構成される。したがって、圧縮時と伸長時で同じ関数g()を使えば、残差„[i]からvel[i]を復元でき、また、関数g()を適当に選ぶことにより、同じデータパターンの„[i]が繰り返して出現することになり、LZ法を用いた場合の圧縮率を改善することができる。
【0039】
【数3】vel[i]＝g(vel[i − 1], vel[i − 2],....., vel[i − T]) + „[i] …(3)
但し、イベントの数をNAとして、i=(T+1),(T+2),.....,NA【0040】次にコントローラ符号生成手段18について説明する。コントローラ符号は図14に示すように、図6に示すコントローラテーブルに登録されたイベントの情報を時間順に並べたものである。各コントローラ符号は、イベントの種類を表すフラグFとパラメータ(データバイト)で構成される。パラメータの個数はイベントの種類により異なる。イベントの種類は大きく分けて「通常イベント」と「連続イベント」の2つのタイプがある。フラグFの最上位ビットが「1」、パラメータの最上位ビットが「0」に設定されているので、SMFと同様のランニングステータス表現(前のイベントと同じ種類のイベントの場合にフラグFを省略すること)が可能になっている。
【0041】ここで、SMFではイベントの種類をあらわすのに1バイトのMIDIステータスが使われている。一般的に使用される値は、8n(hex)、9n(hex)、An(hex)、Bn(hex)、Cn(hex)、Dn(hex)、En(hex)、F0(hex)、FF(hex)のいずれかである(ただし、n=0〜F(hex)、nはチャネル番号である)。「通常イベント」は、上記MIDIステータスからノートオン8n(hex) とノートオフ9n(hex) を除いたものであるが、本発明では前述したようにチャネル番号を表現する必要が無いので、「通常イベント」のフラグの種類は7種類となる。従ってMIDIステータスに比べフラグFは同じ値になる確率が高く、LZ法を用いた場合の圧縮率が高まる。「通常イベント」の符号は、フラグFの後に、SMFのMIDIステータス1バイトを除いたデータバイトを並べたものである。
【0042】また、SMFでは、特定の種類のイベントが一定数以上連続して出現し、各々のイベントのパラメータ値(データバイト)がほぼ一定の規則で変化する部分が存在することが多い。例えば「ピッチホイールチェンジ」イベントが使われている部分である。このイベントは、音符の音程を微妙に変えて音楽的な表現力を高める為のものであり、その性質上パラメータ値の異なる複数イベントが連続して使われることが多い。このようなイベントを「連続イベント」と呼び、このような部分を「連続イベントブロック」と呼ぶ。
【0043】以下の説明では、「連続イベント」の一例として「ピッチホイールチェンジ」を取りあげるが、これに限定されるものではない。SMFの「連続イベントブロック」の一例を図15に示す。この場合、各イベントのパラメータ値が異なる為、SMFの同一データパターンの長さは、(„タイム・1バイト＋ステータス・1バイト)の計2バイトであり、この程度の長さではLZ法による圧縮の効果はほとんど得られない。
【0044】そこで、コントローラテーブルの中で「ピッチホイールチェンジ」が一定数以上連続して出現し、パラメータ値がほぼ一定の規則で変化する領域に対し、以下の処理を施すことによりコントローラ符号を生成する。先ず、「ピッチホイールチェンジ」の数が一定数に満たないような場合は、前述した「通常イベント」として符号化する。そして、(4)式に示すように、連続イベントブロック内のイベントのパラメータp[i] をそれ以前に出現したU個のイベントのパラメータ値p[i − 1] 、p[i − 2] 、.....、p[i − U]を変数とする関数h()と残差„[i]で表す(ただし、p[i − 1] はp[i] の1つ前のパラメータ値、p[i − 2] はp[i] の2つ前のイベントのパラメータ値)。
【0045】連続イベントの符号の構成は図16に示すように、ピッチホイールチェンジが連続していることを意味するフラグFに続き、1番目からU番目までのイベントに対してはパラメータの値そのものである。そして、(U＋1)番目以降のイベントでは、„[i]を時間順に並べたものである。
【0046】
【数4】p[i] = h(p[i − 1], p[i − 2],....., p[i − U]) +„[i] …(4)

ただし、連続イベントブロックのイベント数をNCとしてi = (U + 1), (U + 2), ... NC【0047】関数h()には種々のものが考えられるが、なるべく同じ値の„[i]が繰り返して出現するようなものを選ぶことにより、2次符号生成手段4において効率の良い圧縮が可能となる。一例として(5)式のような関数を使った場合の効果を説明する。これはU＝1であり、1つ前のパラメータとの差分をとることを意味する。

【0048】

【数5】

p[i] = p[i − 1] + „[i] …(5)

ただし、連続イベントブロックのイベント数をNCとしてi = 2,3, ..., NC【0049】この方法によれば、図15に示す領域は図17のようなコントローラ符号に変換される。この場合、2番目以降のイベントのデータが全て同一の「1」になる為、LZ法による圧縮率が高まる。また、コントローラ符号には„タイムが含まれないので、„タイムがイベント毎に異なる場合でも、LZ法における圧縮率低下の影響が少ない。また場合によっては、(4)式の代わりに、イベントの時間情報も変数に使った(6)式のような関数e()を使っても良い。ただし、t[i]はパラメータを求めるイベントの時間、t[i − 1] はその1つ前のイベントの時間である。

【0050】

【数6】p[i] = e(p[i − 1],p[i − 2],.....,p[i − U],t[i],t[i − 1],.....,t[i − U]) +„[i] …(6)

ただし、連続イベントブロックのイベント数をNCとしてi = (U + 1), (U + 2), ... NC【0051】符号配置手段19では、上記の各符号を図18のような領域に配置して1次符号3を生成する。ヘッダは各符号の開始アドレスや長さといった管理情報と前述したチャネルマップを含んでいる。すでに説明したように各符号はSMFに比べ、同一データの出現回数が多く、同一データパターンの長さも長いという性質を持っているが、さらに同一データパターンが近い距離で出現するように工夫をしている。まず同じ種類の符号内で、同じデータ列が出現する確率が高いので、トラック順に同一種類の符号を配置している。また、ノート„符号とコントローラ„符号とデュレーション符号は、全て時間情報であり、性質の異なるノートナンバ符号やベロシティ符号よりも同じデータ列が出現する確率が高いので、これらの距離が近くなるように配置している。

【0052】次に、図19で示した例に戻って、同一データパターンの長さがどの程度改善されるか具体的に検討する。ここで、各々のメロディは、50個のノートオンイベントと50個のノートオフイベントで構成されており、全ての„タイムは1バイトであるとし、全てのイベントは3バイトであると仮定すると、各々のメロディのノートナンバは前述したように全て同じである。

【0053】SMFにおいて、各々のメロディのデータ量は、(1＋3)×50×2＝400バイトである。各々のメロディの„タイムとベロシティが全て同じならば、同一データパターンの長さは400バイトになる。しかし両メロディ間の全ての„タイムとノートオンのベロシティが異なっているとすると、SMFで同一データパターンの最大長は、ノートオフステータス、ノートナンバ、ベロシティの並びの3バイトである。この程度ではLZ法の圧縮はほとんど効果がない。

【0054】一方、本発明では、„タイム、ノートナンバ、ベロシティを分離して符号化しているので、少なくともノートナンバ符号の中で50バイトの長さの同一データパターンが出現する。また前述したようにSMFのベロシティが全く異なる場合でも、ベロシティ符号の中では同一データパターンが出現することが多い。従ってLZ法による圧縮率は明らかに改善される。以上の説明から分かるように1次符号3は、SMFの持つ音楽的な情報量を全く落とすことなくデータ量が削減されていると同時に、SMFに比べ同一のデータパターンの長さが長く、出現回数が多く、しかもそれらが近い距離で出現するようの性質を持っているので、2次符号生成手段4において効果的な圧縮を行うことができる。また、この1次符号3そのものも、かなり圧縮されたデータ量となっているので、この1次符号3を直接出力するようにしてもよい。

【0055】2次符号生成手段4においては、1次符号生成手段2の出力3に対して、LZ法による圧縮を行う。LZ法は、gzip、LHAといった圧縮プログラムで広く使われている手法である。これは入力データの中から、同一のデータパターンを捜し、もし存在すれば、(過去に出現したパターンへの距離、パターンの長さ)という情報に置き換えて表現することでデータ量を削減する。例えば、"ABCDEABCDEF"というデータを、"ABCDE"が繰り返しであるので、"ABCDE(5, 5)F"という情報に置き換える。なお、圧縮符号(5, 5)は5文字戻って5文字コピーすることを表す。

【0056】処理の概要は次のようになる。2次符号5の生成は、処理位置を1次符号3の先頭から順次移動させて行う。処理位置のデータパターンが、それ以前の一定の範囲内のデータパターンと一致する場合は、処理位置からそのデータパターンまでの距離と、一致したデータパターンの長さを2次符号5として出力し、処理位置を2つ目のデータパターンの終わりに移動させ、処理を続ける。処理位置のデータパターンが、それ以前の一定の範囲内のデータパターンと一致しなければ、1次符号3をコピーして2次符号5として出力する。

【0057】以上の説明から明らかなように、2つの同一のデータ領域が大きいほど、圧縮率は高くなる。また同一のデータ領域の距離は一定範囲内である必要がある。前述したように楽曲の中では、同じようなメロディーが繰り返し使われるが、SMFのままでそれらのデータを比較すると、完全に同一のデータ列の繰り返しであることは少なく、むしろノートナンバは同じであるがベロシティは異なるといったように、どこか一部分異なっていることが多い。

【0058】一方，本発明では、性質の同じデータを独立した領域にまとめると同時に、各領域で同一のデータがなるべく多く出現するような処理を行ない、さらに性質の近い領域どうしをなるべく近くに配置することにより、LZ法の圧縮率が高まるので、最終的な2次符号5は十分容量の小さなものになる。なお、以上詳述したフォーマット並びに処理手順は一例であり、その主旨を逸脱しない範囲において種々の変更を加えることができる。また、演奏情報としてSMFを例に取ったが、SMFに限らずこれに類似の演奏情報に対して本発明を適用してデータ容量を効率よく削減することができる。

【0059】次に、図20～図28を参照して上記の1次符号3又は2次符号5を復号するための演奏情報復号装置について説明する。図20は演奏情報復号装置を示すブロック図、図21は図20の2次符号復号手段の処理を説明するためのフローチャート、図23は図22のトラック復号処理を詳しく説明するためのフローチャート、図22は図20の1次符号復号手段の処理を説明するためのフローチャート、図24は図23のノートイベント復号処理を詳しく説明するためのフローチャート、図25は図24のノートイベント復号処理により復元されたノートオンイベントを示す説明図、図26は図24のノートイベント復号処理により復元されたノートオフキューを示す説明図、図27は図23のコントローライベント復

号処理を詳しく説明するためのフローチャート、図28は図27の処理により復元されたコントローライベントを示す説明図である。

【0060】図20では圧縮処理とは逆に、LZ法で圧縮された入力データ21が2次符号復号手段23により音程と、音の強さと、音の長さとその他の情報に分離された1次符号3に復号され、次いで1次符号復号手段24により元の音符(出力データ25)に復元される。制御手段26はスイッチ22により、入力データ21が図1に示す2次符号5である場合に2次符号復号処理に続き1次符号復号処理を行うように制御し、入力データ21が図1に示す1次符号3である場合に1次符号復号処理のみを行うように制御する。

【0061】ここで、2次符号5であるか又は1次符号3であるかの判定は、キーボード、マウス、ディスプレイ等の図示しない入出力装置を使用してオペレータが指定してもよいし、圧縮された情報に対して符号化方法の種類を示す情報を符号化時に付加し、復号時にこの情報を自動的に判別するようにしてもよい。

【0062】次に、図21を参照して2次符号復号手段23の復号処理を説明する。入力データ11(2次符号5)を先頭から読み込み(ステップS101)、次いで圧縮データの部分であるかすなわちABCDE(5, 5)の「ABCDE」の部分(=非圧縮データ)であるか又は「(5, 5)」の部分(=圧縮データ)であるかを判定する(ステップS102)。

【0063】そして、圧縮データの部分である場合には過去に出現した同一パターンを参照してそれをコピーして出力し(ステップS103)、他方、非圧縮データの部分である場合にはそのまま出力する(ステップS104)。以下、入力データ11を全て復号するまでこの処理を繰り返すと(ステップS105→S101)、図18に示すような配置の1次符号3が復元される。

【0064】次に、図22及び図18に示す1次符号3を参照して1次符号復号手段24の復号処理を説明する。先ず、1次符号3のヘッダを読み込む(ステップS111)。ヘッダには総トラック数N、ノート„符号からコントロール符号までの各符号領域の先頭アドレス、チャネルマップ、時間分解能等の情報が符号化の際に記録されているので、このヘッダ情報に基づいてSMFのヘッダを作成して出力する(ステップS112)。

【0065】次にトラック番号iを「1」にセットし(ステップS113)、図23に詳しく示すトラック復号処理を行う(ステップS114)。次いでトラック番号iが総トラック数Nより小さいか否かをチェックし(ステップS115)、もし小さければトラック番号iを1つインクリメントし(ステップS116)、ステップS114に戻ってトラック復号処理を繰り返す。そして、ステップS115においてトラック番号iが総トラック数Nより小さくない場合にこの1次符号復号処理を終了する。

【0066】図23に詳しく示すトラック復号処理では、先ず、処理で使用する変数を初期化する(ステップS121)。具体的は、処理中のノートイベントの番号を示す変数jを「1」にセットし、処理中のコントローライベントの番号を示す変数kを「1」にセットし、ノート終了フラグとコントローラ終了フラグをクリアする。ここで、ノート終了フラグは処理トラックの全てのノートイベントの復号が終了したことを示し、コントローラ終了フラグは処理トラックの全てのコントローライベントの復号が終了したことを示す。

【0067】次に処理トラック番号iのノート„符号の最大公約数„Tsnと、コントローラ„符号の最大公約数„Tscとデュレーション符号の最大公約数Dsを読み出す(ステップS122)。そして、j番目のノート„符号„Tan[j]とk番目のコントローラ„符号„Tac[k]を読み出し、(7)式のように各々最大公約数„Tsn、„Tscを乗じて„Tn[j]、„Tc[k]を算出する(ステップS123)。

„Tn[j]=„Tan[j] ×„Tsn„Tc[k]=„Tac[k] ×„Tsc …(7)

【0068】さらに、(8)式のようにトラックの先頭を基準とした時刻Tn[j]、Tc[k]に変換する(ステップS124)。

Tn[j]=Tn[j-1]+„Tn[j]Tc[k]=Tc[k-1]+„Tc[k]ただし、Tn[0]=Tc[0]=0 …(8)

なお、ステップS123、S124では、ノート終了フラグがセットされている場合には„Tn[j]、Tn[j]の算出は行わず、また、コントローラ終了フラグがセットされている場合には„Tc[k]、Tc[k]の算出は行わない。

【0069】次に、出力すべきノートオフイベントの有無をチェックし(ステップS125)、出力すべきデータが有る場合にはSMFとしてノートオフイベントを出力する(ステップS126)。なお、ステップS125、S126については後述(図24のステップS144)する。次に、復号処理の選択を行う。先ず、コントローラ終了フラグをチェックし(ステップS127)、セットされている場合には図24に詳しく示すノートイベント復号処理を行う(ステップS128)。

【0070】コントローラ終了フラグがセットされていない場合にはノート終了フラグをチェックし(ステップS129)、セットされている場合には図27に詳しく示すコントローライベント復号処理を行う(ステップS130)。2つのフラグが共にセットされていない場合にはTn[j]とTc[k]を比較し(ステップS131)、Tn[j]が小さければノートイベント復号処理(ステップS128)を、そうでなければコントローライベント復号処理(ステップS130)を行う。

【0071】ノートイベント復号処理の後には、処理トラックNの全てのノートイベントを処理したか否かをチェックし(ステップS132)、処理が終了している場合にはノート終了フラグをセットし(ステップS133)、ステップS138に進み、そうでなければ変数jを1つインクリメントし(ステップS134)、ステップS123に戻る。また、コントローライベント復号処理の後には、処理トラックNの全てのコントローライベントを処理したか否かをチェックし(ステップS135)、処理が終了している場合にはコントローラ終了フラグをセットし(ステップS136)ステップS138に進み、そうでなければ変数kを1つインクリメントし(ステップS137)、ステップS123に戻る。

【0072】ステップS138ではノート終了フラグとコントローラ終了フラグの両方がセットされているか否かをチェックし、両方がセットされている場合にはこのトラック復号処理を終了し、そうでなければステップS123に戻ってこのトラック復号処理を繰り返す。

【0073】図24に詳しく示すノートイベント復号処理では、先ず、j番目のノートナンバ符号„[j]を読み取り、圧縮処理において使用した関数f()を用いて(9)式に従ってノートナンバnum[j]を算出する(ステップS141)。

【0074】

【数7】

num[j]=f(num[j-1],num[j-2],.....,num[j-S]) + „[j] (j>S) num[j]=„[j] (j≦S) ただし、S は関数f()の変数の個数 …(9)

【0075】同様に、j番目のベロシティ符号„[j]を読み取り、圧縮処理において使用した関数g()を用いて(10)式に従ってベロシティvel[j]を算出する(ステップS142)。

【0076】

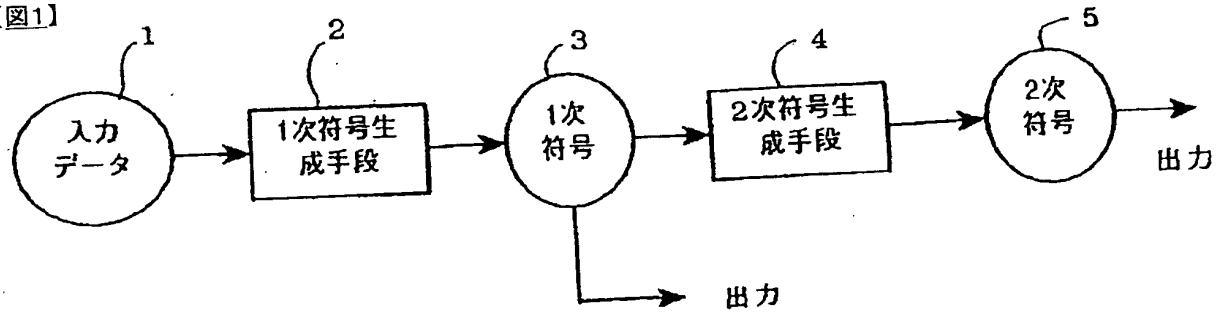
【数8】
vel[j]＝g(vel[j−1],vel[j−2],,,,,,,vel[j−T]) + „[j] (j＞T) vel[j]＝„[j] (j≦T) ただし、T は関数g()の変数の個数 …(10)
【0077】次いでTn[j]、num[j]、vel[j]を用いて図25に示すようなノートオンイベントを出力する(ステップS143)。なお、SMTの„タイム„ Tは、Tn[j]の直前に出力したイベントの時刻Tb を使って式(11)に従って求め、出力する。
„T＝Tn[j]−Tb …(11)
図25に示すノートオンイベントにおけるステータスバイトの上位4ビットはノートオン「9(hex)」を表し、下位4ビットはチャネルマップから得られる番号が続く。このステータスバイトの後にはノートナンバとベロシティの各バイトが続く。
【0078】次にノートオフイベントの登録を行う(ステップS144)。具体的にはデュレーション符号Da[j]を読み取って、(12)式に従いノートオフイベントの時刻Toff を算出し、この時刻Toff とノートナンバnum[j]を図26に示すようなノートオフキューに登録する。このノートオフキューでは、使用されているエントリの数を保持するとともに、ノートオフ時刻Toff が先頭から小さい順に並ぶように管理される。
Toff[j]＝Da[j]×Ds ＋Tn[j] …(12)
【0079】前述した図23のステップS125においては、Tn[j]とTc[k]の内の値が小さいほうTm をノートオフキューの先頭のToff[n](n＝1〜エントリ総数N)から順に比較する。Toff[n]＜Tm であるエントリがあればステップS126に進み、ノートオフイベントを出力する。ステップS126では、前述したノートオフイベントをSMFとして出力する。
【0080】次に図27を参照してコントローライベント復号処理を詳しく説明する。この処理では図28に示すように„タイム、ステータス及びパラメータより成るコントローライベントが復元され、先ず、Tc[k]と直前に出力したイベントの時刻Tb を使って(13)式に従ってSMTの„タイム„ Tを求め、出力する(ステップS151)。
„T＝Tc[k]−Tb …(13)
【0081】次にコントローラ符号領域からイベントの種類を表すイベントフラグF[k] を読み取り、F[k] が「通常イベント」であるか、「連続イベント」であるか又は「ランニングステータス」であるかを判定する(ステップS152)。ここで、連続イベントブロック内では、図示省略16に示すように、2番目以降のイベントはイベントフラグが省略された「ランニングステータス」状態で記録されている。
【0082】F[k] が「通常イベント」である場合には、処理イベントの連続イベントブロック内における順番を示す変数mを「0」にリセットし(ステップS153)、次いでチャネルマップを参照してSMFのステータスバイトを作成して出力する(ステップS154)。さらにイベントの種類に応じて必要なバイト数をコントローラ符号領域から読み出し、この読み出した値がSMFのパラメータ(データバイト)であるのでこれを出力する(ステップS155)。
【0083】F[k] が「連続イベント」である場合には、連続イベントブロック内における順番を示す変数mを「1」にセットし(ステップS156)、次いでチャネルマップを参照してSMFのステータスバイトを作成して出力する(ステップS157)。なお、m≧2の場合のステータスバイトはmが「1」の場合のステータスバイトを利用する。そして、この「連続イベント」の場合には、パラメータ符号„[m] を読み出し、圧縮処理と同じ関数h()を使い、(14)式に従ってSMFのパラメータp[m] を作成し、出力する(ステップS158)。
【0084】
【数9】
p[m] ＝h(p[m−1],p[m−2],,,,,,p[m−U] + „[m] (m＞U)
p[m] ＝„[m] (m≦U)
ただし、Uは関数h()の変数の個数 …(14)
【0085】F[k] が「ランニングステータス」である場合には、変数mの値をチェックし(ステップS159)、mが「0」より大きければmを1つインクリメントし(ステップS160)、「連続イベント」側のステップS157に進む。他方、mが「0」であれば「通常イベント」側のステップS154に進む。

図の説明

【図面の簡単な説明】
【図1】本発明に係る演奏情報圧縮装置の一例を示すブロック図である。
【図2】図1の1次符号生成手段の一例を詳細に示すブロック図である。
【図3】図2のチャネル分離手段により作成されるチャネルマップを示す説明図である。
【図4】図2の解析手段の処理を説明するためのフローチャートである。
【図5】図2の解析手段により作成されるノートテーブルを示す説明図である。
【図6】図2の解析手段により作成されるコントローラテーブルを示す説明図である。
【図7】音符を表現するSMFの„タイムと本実施例のデュレーションの関係を示す説明図である。
【図8】図2のノート„符号生成手段の処理を説明するためのフローチャートである。
【図9】図2のノート„符号生成手段により生成されるノート„符号を示す説明図である。
【図10】図2のデュレーション符号生成手段の処理を説明するためのフローチャートである。
【図11】図2のデュレーション符号生成手段により生成されるデュレーション符号を示す説明図である。
【図12】図2のノートナンバ符号生成手段により生成されるノートナンバ符号を示す説明図である。
【図13】図2のベロシティ符号生成手段により生成されるベロシティ符号を示す説明図である。
【図14】図2のコントローラ符号生成手段により生成されるコントローラ符号を示す説明図である。
【図15】SMFの連続イベントブロックを示す説明図である。
【図16】本実施例の連続イベントブロックを示す説明図である。
【図17】図16の連続イベントブロックの効果を示す説明図である。
【図18】図2の符号配置手段により並べ替えられた1次符号を示す説明図である。
【図19】SMFのフォーマットを示す説明図である。
【図20】演奏情報復号装置を示すブロック図である。
【図21】図20の2次符号復号手段の処理を説明するためのフローチャートである。
【図22】図20の1次符号復号手段の処理を説明するためのフローチャートである。
【図23】図22のトラック復号処理を詳しく説明するためのフローチャートである。
【図24】図23のノートイベント復号処理を詳しく説明するためのフローチャートである。
【図25】図24のノートイベント復号処理により復元されたノートオンイベントを示す説明図である。
【図26】図24のノートイベント復号処理により復元されたノートオフキューを示す説明図である。
【図27】図23のコントローライベント復号処理を詳しく説明するためのフローチャートである。
【図28】図27の処理により復元されたコントローライベントを示す説明図である。
【符号の説明】
1 入力データ
2 1次符号生成手段
3 1次符号
4 2次符号生成手段
5 2次符号
11 チャネル分離手段
12 解析手段
13 ノート„符号生成手段
14 コントローラ„符号生成手段
15 デュレーション符号生成手段
16 ノートナンバ符号生成手段
17 ベロシティ符号生成手段
18 コントローラ符号生成手段
19 符号配置手段
21 入力データ
22 スイッチ
23 2次符号復号手段
24 1次符号復号手段
25 出力データ
26 制御手段

図面

【図1】

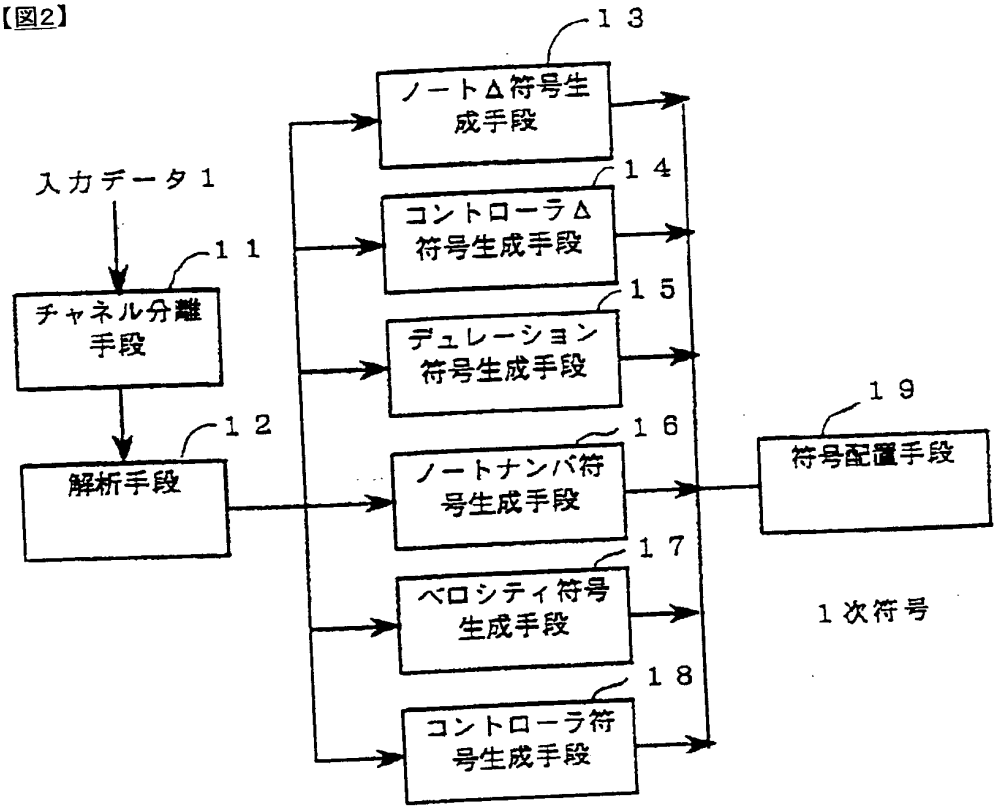


【図25】

ノートオンイベント

Δタイム	ステータス (9x hex)	ノートナンバ	ベロシティ

【図2】



【図3】

チャネルマップ

トラック	チャネル番号
1	2
2	3
3	5
4	1

【図5】

ノートテーブル

	時間	ノートナンバ	ベロシティ	デュレーション	ノートオフ参照	ΔT
ノート1	480	60	80	240	1	
ノート2	640	62	80		0	
ノートNA						

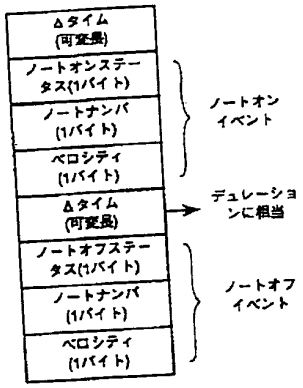
【図6】

コントローラテーブル

	時間	データ
イベント1		
イベント2		
イベントNB		

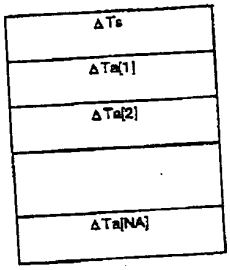
【図7】

SMF



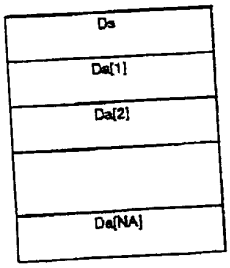
【図9】

ノートΔ符号

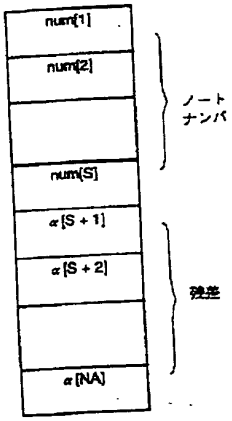


【図11】

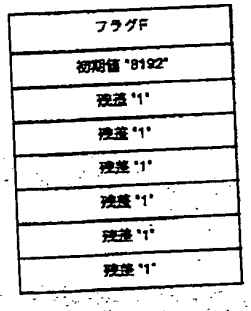
デュレーション符号



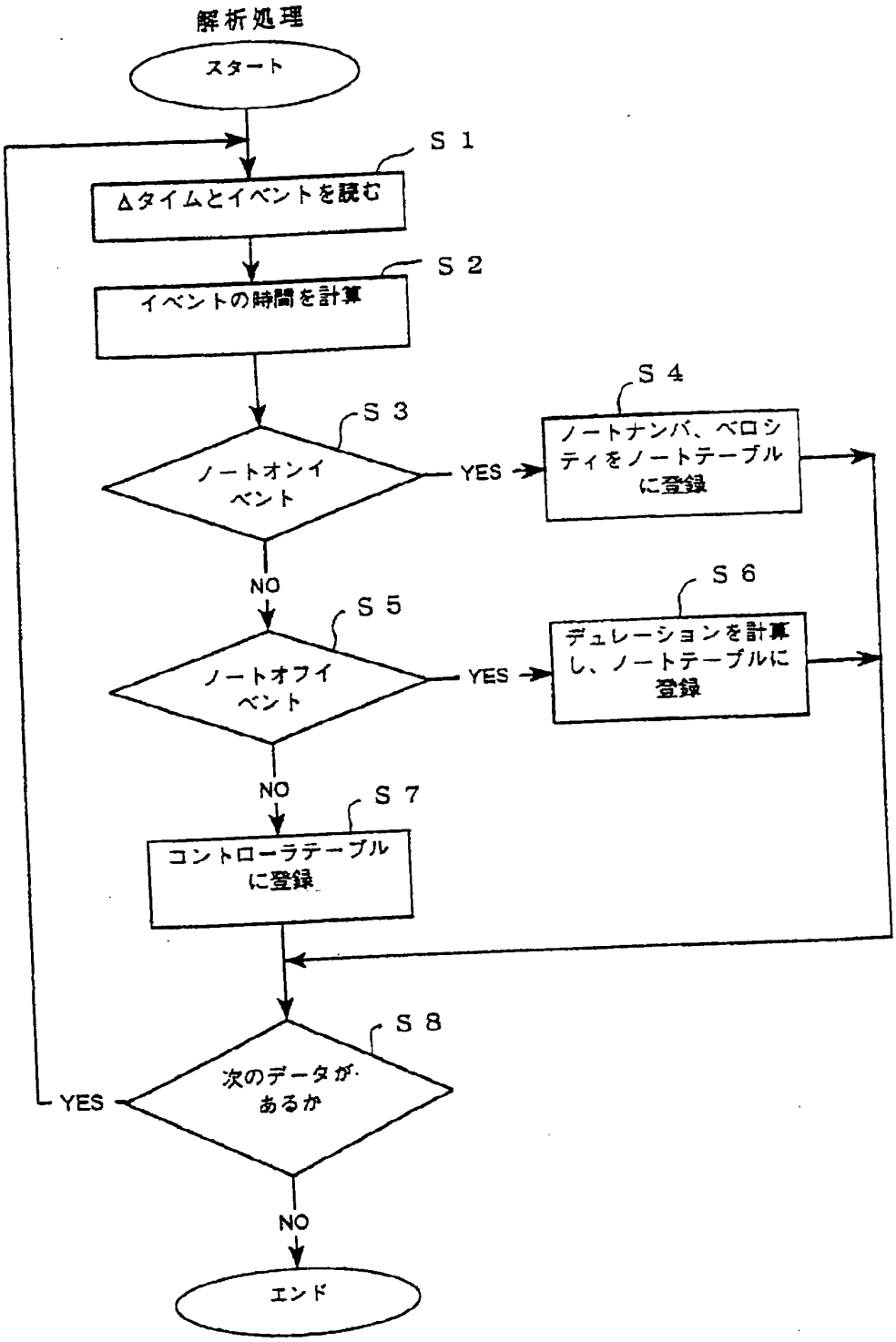
【図12】
ノートナンバ符号



【図17】
コントローラ符号

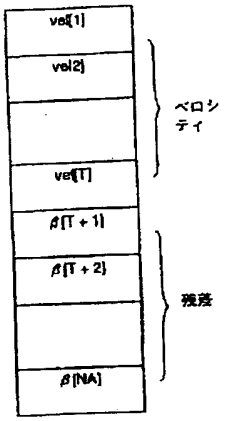


【図4】


解析処理
スタート
S 1
Δタイムとイベントを読む
S 2
イベントの時間を計算
S 3
ノートオンイベント
YES
S 4
ノートナンバ、ベロシティをノートテーブルに登録
NO
S 5
ノートオフイベント
YES
S 6
デュレーションを計算し、ノートテーブルに登録
NO
S 7
コントローラテーブルに登録
S 8
次のデータがあるか
YES
NO
エンド

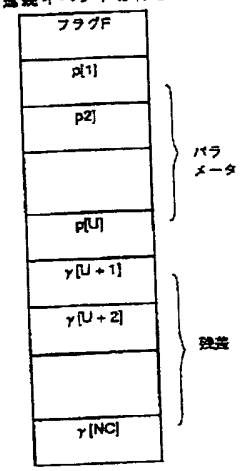
【図13】

ベロシティ符号

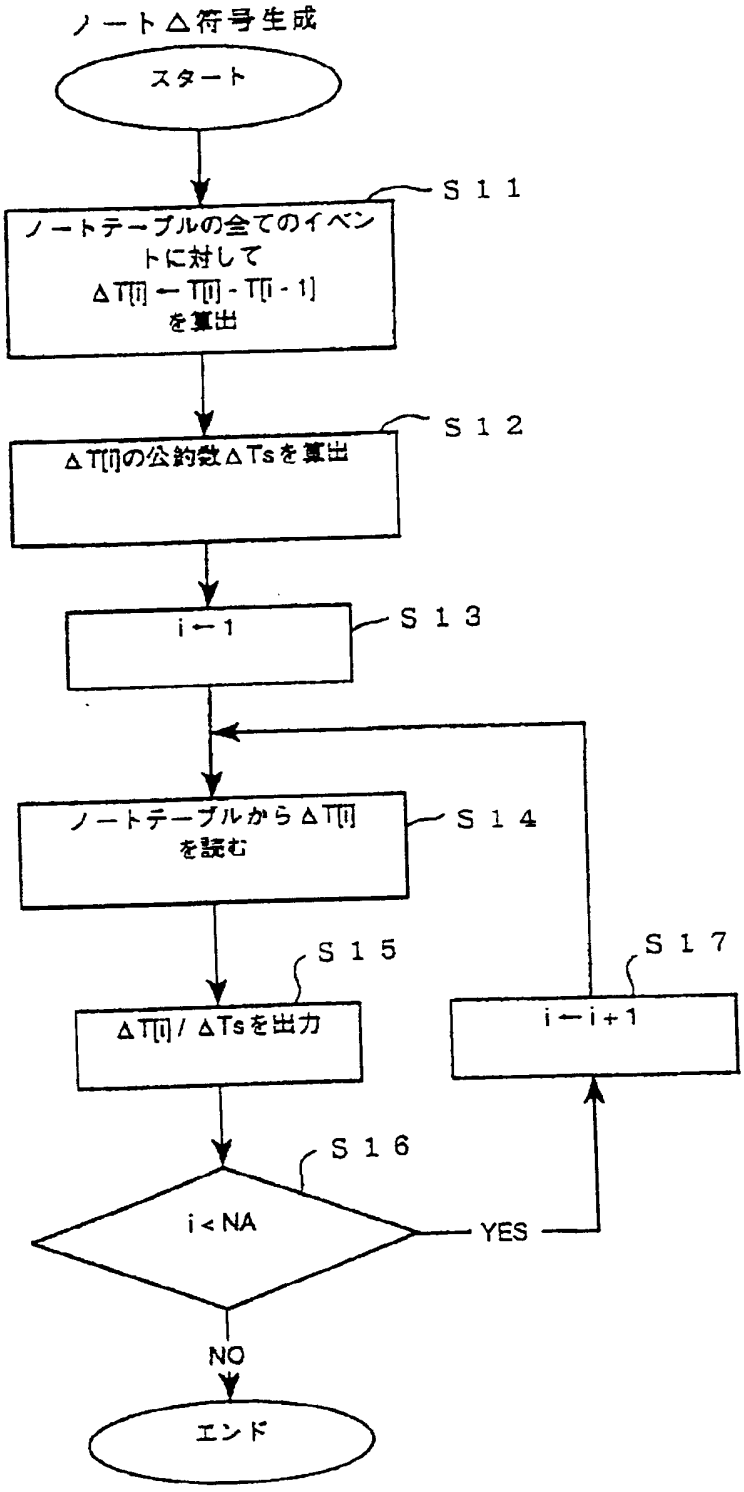


【図16】

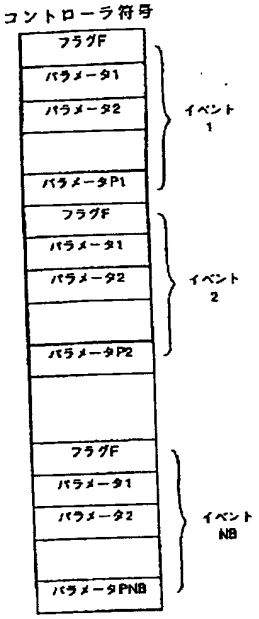
連続イベントの符号



【図8】


ノート△符号生成
スタート
S 1 1
ノートテーブルの全てのイベントに対して
ΔT[i] ← T[i] - T[i - 1]
を算出
S 1 2
ΔT[i]の公約数ΔTsを算出
i ← 1
S 1 3
ノートテーブルからΔT[i]を読む
S 1 4
S 1 5
ΔT[i] / ΔTsを出力
S 1 7
i ← i + 1
S 1 6
i < NA
YES
NO
エンド

【図14】

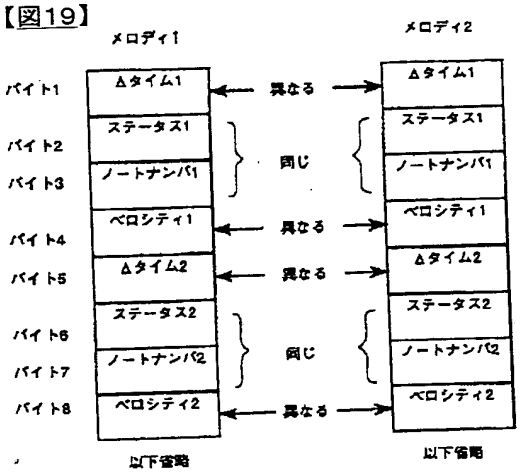


【図15】

連続イベントブロック

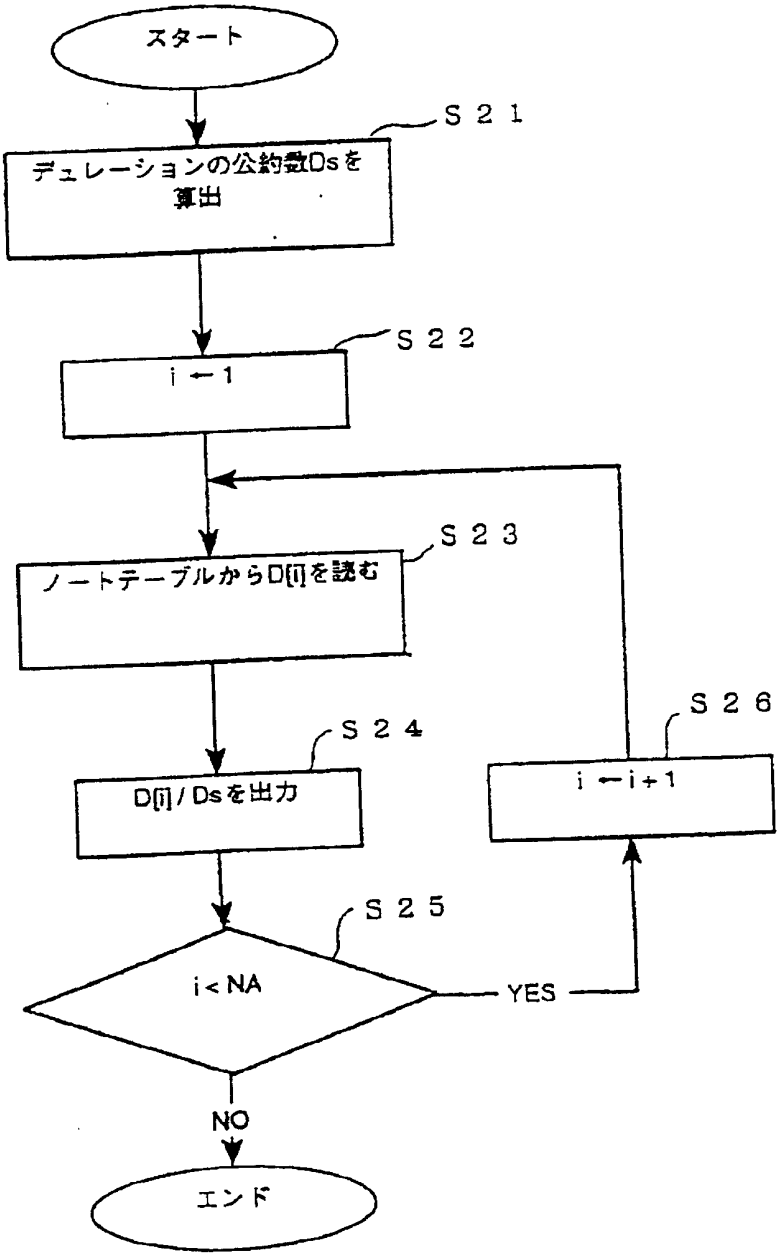
Δタイム	ステータス	パラメータ
10	224(ピッチホイールチェンジ)	8192
20	224(ピッチホイールチェンジ)	8193
20	224(ピッチホイールチェンジ)	8194
10	224(ピッチホイールチェンジ)	8195
30	224(ピッチホイールチェンジ)	8196
10	224(ピッチホイールチェンジ)	8197
20	224(ピッチホイールチェンジ)	8198

【図19】

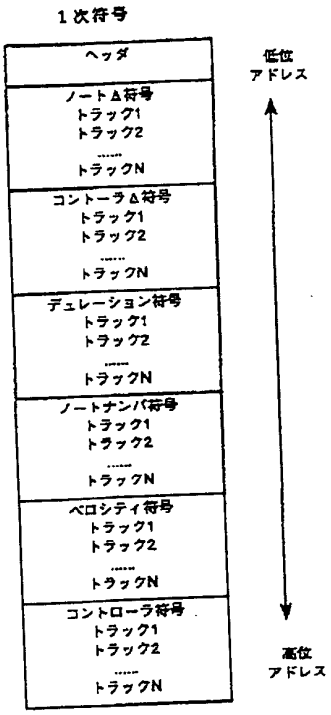


【図10】

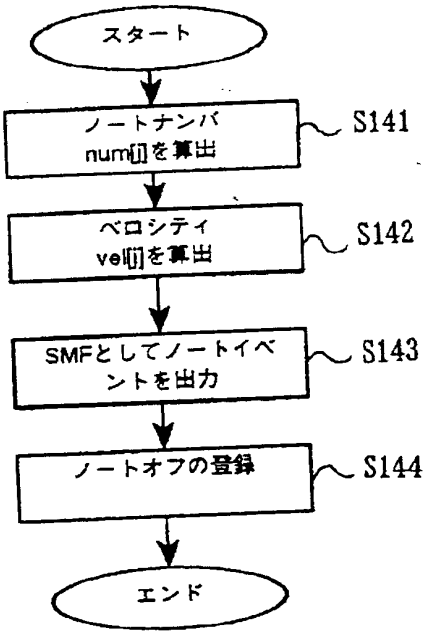
デュレーション符号生成



【図18】



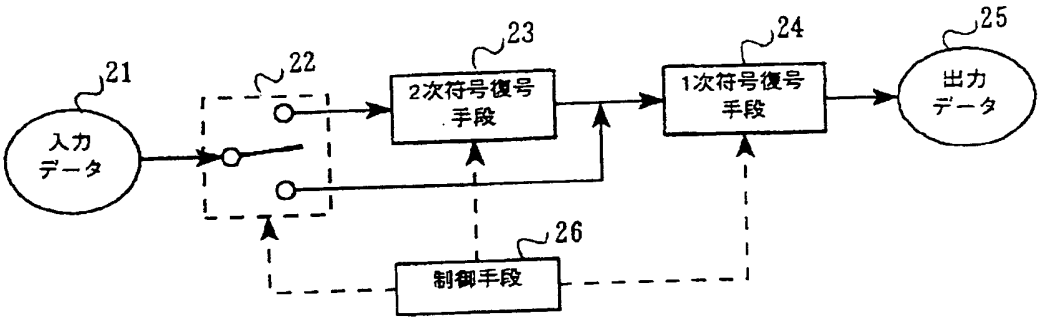
【図24】
ノートイベント復号処理



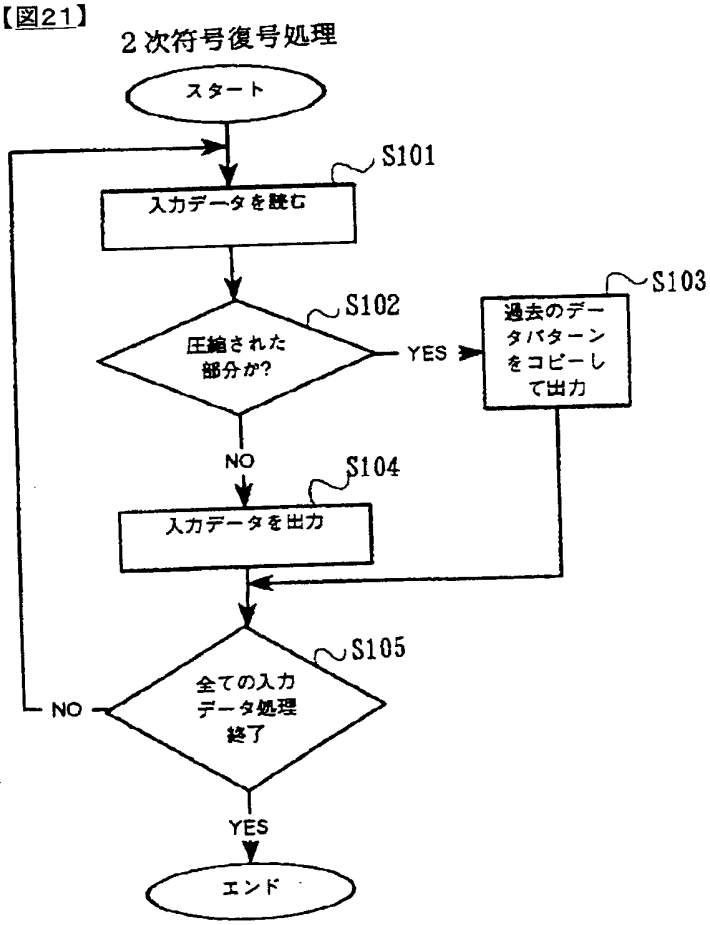
【図26】
ノートオフキュー

	Toff	ノートナンバ
エントリ1	1000	84
エントリ2	1100	54

【図20】

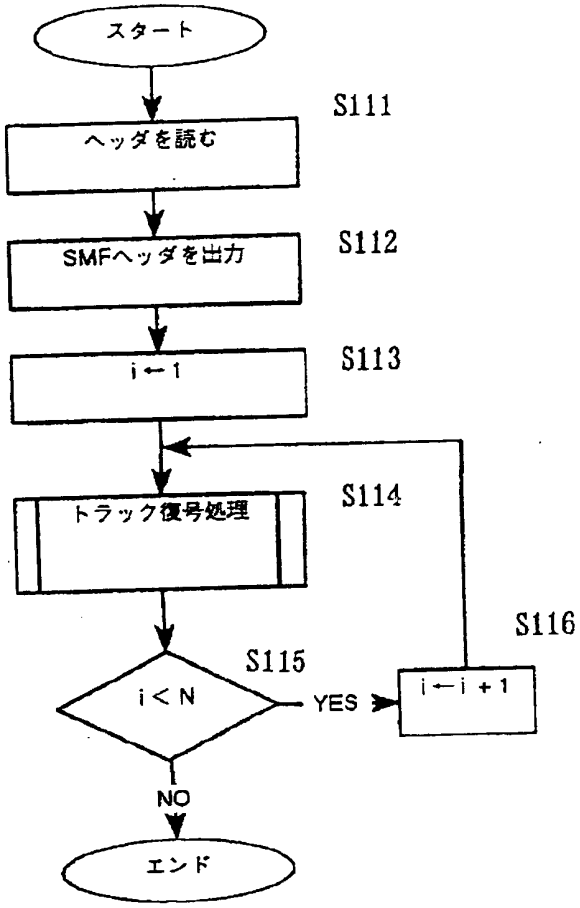

21
入力データ
22
23
2次符号復号手段
24
1次符号復号手段
25
出力データ
26
制御手段

【図21】

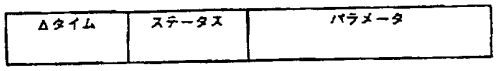

2次符号復号処理
スタート
S101
入力データを読む
S102
圧縮された部分か?
YES
S103
過去のデータパターンをコピーして出力
NO
S104
入力データを出力
S105
全ての入力データ処理終了
NO
YES
エンド

【図22】

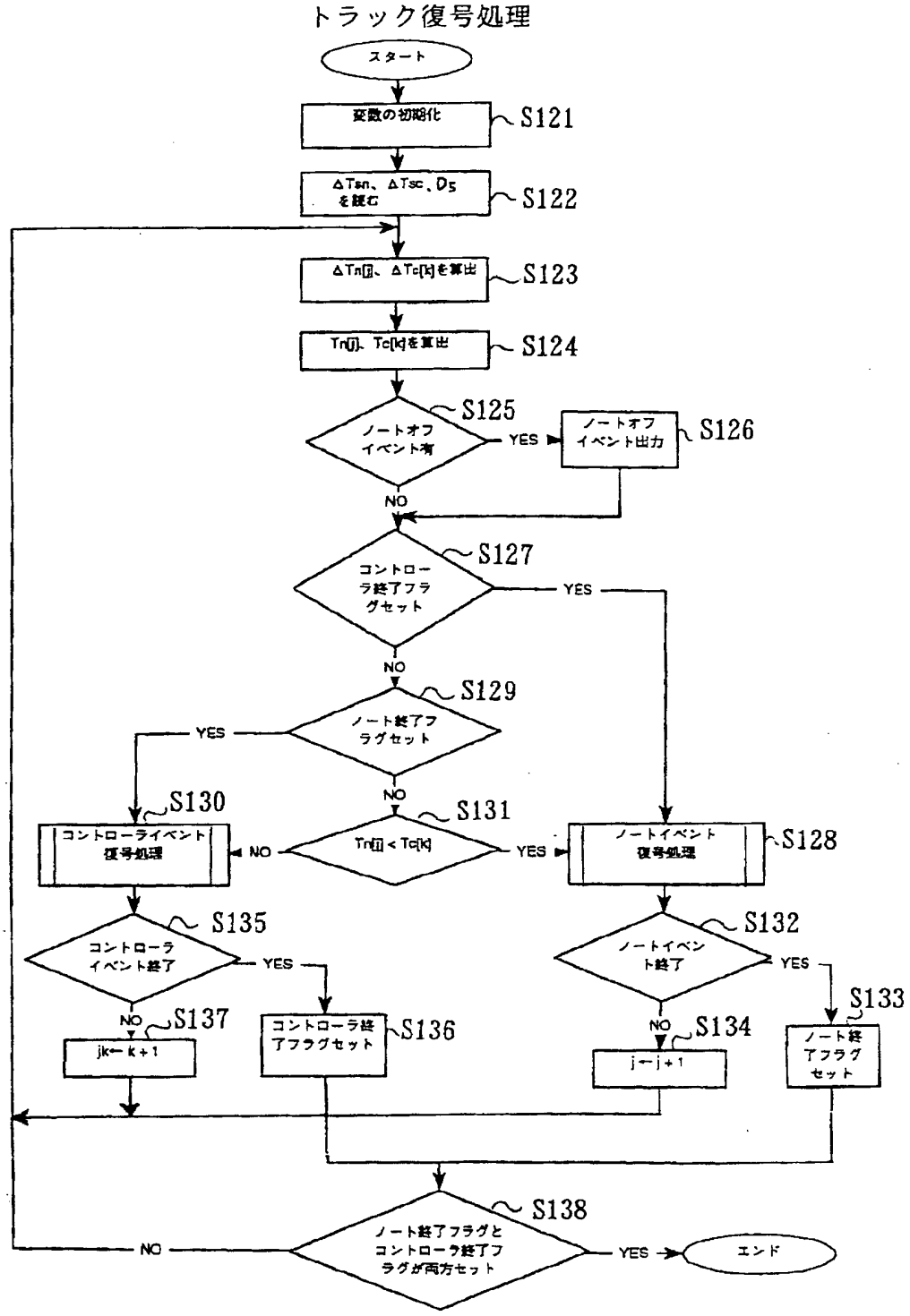
1次符号復号処理



【図28】
コントローライベント



【図23】


トラック復号処理
スタート
変数の初期化
S121
ΔTsn、ΔTsc、D5を読む
S122
ΔTn[j]、ΔTc[k]を算出
S123
Tn[j]、Tc[k]を算出
S124
ノートオフイベント有
S125
YES
ノートオフイベント出力
S126
NO
コントローラ終了フラグセット
S127
YES
NO
ノート終了フラグセット
S129
YES
NO
コントローライベント復号処理
S130
Tn[j] < Tc[k]
S131
NO
YES
ノートイベント復号処理
S128
コントローライベント終了
S135
YES
ノートイベント終了
S132
YES
NO
jk←k+1
S137
コントローラ終了フラグセット
S136
NO
j←j+1
S134
ノート終了フラグセット
S133
ノート終了フラグとコントローラ終了フラグが両方セット
S138
NO
YES
エンド

【図27】

コントローライベント復号処理